Authentication and Privacy with Application to Pervasive Computing

Michael Østergaard Pedersen

PhD Dissertation

Department of Computer Science
University of Aarhus
Denmark

Authentication and Privacy with Application to Pervasive Computing

A Dissertation
Presented to the Faculty of Science
of the University of Aarhus
in Partial Fulfilment of the Requirements for the
PhD Degree

by
Michael Østergaard Pedersen
30th May 2008

Abstract

Knowing who you are interacting with, i.e. authenticating their identity, is crucial to achieve security in almost all scenarios, since authentication is the basis for authorization, access control, non-repudiation, and auditing. However, technologies used for identification and authentication are often plagued by privacy problems, since they by their nature, often leave digital traces behind, that allows one to uniquely identity a user across different domains, or link a user's identity to his actions. Fortunately there do exist technologies that can give us the best of both worlds: Privacy, while still being able to authenticate the credentials needed for a given task.

In this thesis we examine how authentication can be applied to different scenarios in the pervasive computing domain, where each scenario have their own specific requirements such as privacy, usability, efficiency, or a combination. More specifically: 1) We examine the issue of privacy and authentication related to RFID tags, and show that some tradeoffs between privacy and efficiency are unavoidable. 2) We propose an authentication scheme that allows users to remain anonymous, as long as they do not try to cheat the system by giving away their credentials. 3) We consider the importance of usability to authentication, by 3a) designing a user friendly login system, supporting work in a hospital environment, and 3b) designing and implementing an anti-theft policy for pervasive computing devices where the key property is to authenticate the user towards a system of devices in his own home. 4) Short signatures are important in many applications, especially in the pervasive computing realm, but unfortunately most short signatures are slow to verify. We propose batch verifiers for a number of signature schemes, we propose a new signature scheme with very short signatures, for which batch verification for many signers is also highly efficient, and finally we implement some of the schemes and provide an in depth performance analysis.

Acknowledgements

First of all I want to thank my supervisor Ivan Damgård for his help and support as well as his enthusiasm and many valuable contributions to this work. I am also grateful to Jakob Illeborg Pagter and Klaus Marius Hansen for giving me the opportunity to do my PhD here, as part of the eu-DOMAIN project.

I thank all my co-authors Jakob Bardram, Jan Camenisch, Ivan Damgård, Kasper Dupont, Matthew Green, Anna Lisa Ferrara, Susan Hohenberger, Rasmus Kjær, Jakob Illeborg Pagter and Torben Pedersen for their help and many inspiring discussions. Also thanks to Serge Fehr and Anna Lysyanskaya for taking time out of their busy schedule to review this thesis.

A special thanks to Jan Camenisch for hosting me during my abroad stay at IBM Research in Zürich. It was a very nice and memorable experience. I also thank all the people I met there, especially Susan Hohenberger for great conversations of both scientific, and not so scientific, nature.

Studying here at the University of Aarhus has been a great experience, caused by both the excellent environment for cryptographic research, that members of the cryptology group provides, but also because of all the friends that have crossed my path over the many years I have been here. Unfortunately there are too many to mention all of them, but rest assured that I have not forgotten any of you. Thank you all for having a great time here with me.

It is always a pleasure to work in an environment where everything just seems to work. Of course we all know that nothing just works by itself, so I would like to thank the people who makes it seem like it does: The current and former secretaries Dorthe, Ellen, Hanne, Karen and Lene, the technical staff who make everything from the computers to the coffee machines work, and the cleaning personnel who have been making my office a nice place to return to every morning.

Last, but not least, I want to express my gratitude to my family for their love and support, and for bearing with me during those periods when work took a very big part of my time.

Michael Østergaard Pedersen,
Århus, 30th May 2008.

Contents

Chapter 1

Introduction

Knowing who you are interacting with, i.e. authenticating their identity, is crucial to achieve security in almost all scenarios, since authentication is the basis for authorization, access control, non-repudiation, and auditing.

Traditional authentication of a human towards a computer system, has been done by first identifying the user with a username, and then authenticating him by his password. However, that approach does not always work when we consider authentication in a broader sense. In a world where computers are everywhere, and in every form imaginable, we face challenges of authenticating not just a user using a traditional personal computer, but users towards various sorts of computing devices, devices towards each other, the origin of messages, etc. Clearly in most of these cases, tradition username and password authentication is not applicable.

Probably no one will argue against the claim, that technology is playing a bigger and bigger role in our daily lives. We no longer have the option of choosing not to use technology, and that puts additional requirements on all technology we interact with, especially the one used for authentication. It must be easy to use by anyone, yet still remain secure so we can trust it to protect our most sensitive data. It must also be applied in a way that is consistent with what we expect from it, meaning that the use of technology should not have unexpected (negative) consequences for us as users.

Especially technologies used for identification and authentication are often plagued by privacy problems. The purpose of these technologies are by their nature to verify whether or not a user with a specific credential is present, but if we don't control who gets access to that information, or how that information is used, we create privacy problems for the users of the very same technology. Fortunately there do exist technologies that can give us the best of both worlds: Privacy, while still being able to authenticate the credentials needed for a given task.

In this thesis we examine how authentication can be applied to different scenarios in the pervasive computing domain, where each scenario have their own specific requirements such as privacy, usability, efficiency, or a combination.

The rest of this chapter is an introduction to the areas we touch in this thesis. Most of the results are motivated by application to the area of pervasive computing, so we start by giving an introduction to pervasive computing in

Section 1.1. Following that, we examine privacy in Section 1.2. We start by giving an introduction to what privacy is and why we should care about it, and then we shall examine at how privacy violations occur. This will be useful in especially Chapter 3 and 4, where we apply our techniques to solve privacy problems in the real world. Then we relate privacy to the area of pervasive computing. In Section 1.3 we introduce the concept of authentication, and relates it to related work in that area. Finally we give an outline of the rest of this thesis in Section 1.4.

1.1 Pervasive Computing

The term *pervasive computing* covers the vision of computers everywhere, and is characterized by computers being spread throughout the environment, and yet gracefully integrated with it. The effect of this, is that users, devices, and services will be more mobile, information appliances are available when we need them, and communication is made easier between individuals, between individuals and devices, and between devices themselves.

To get a better feeling for the way pervasive computing will change our daily lives in the future, consider the following list of pervasive computing technologies, taken from the report *Things that Think* [145]:

Interactive Spaces In addition to consumer electronics in private homes, which we already have a lot of, this includes things such as swivel chairs in offices that remember height settings for each individual, intelligent fridges that know whether or not the milk is too old, and under-floor heating that is automatically turned on when the forecast is for cold weather.

Clothes Jackets with built-in music players, sports clothing with built-in heart rate monitors, and glasses with built-in computer monitors. Other examples could be intelligent work clothes for groups such as firemen.

Healthcare Intelligent bandages that can report how the injury is doing, video conferences with doctors, so patients can be treated at home, doors that open automatically, so the rescue services can enter the home of a patient with insulin shock and much more.

Retail, Trade and Production There are a large number of examples in the retail trade, such as shoplifting prevention systems that work by the product triggering an alarm when it leaves the shop, automatic stock-taking, and checkouts without cashiers, where the shopping trolley is pushed through a scanner that registers the products. Researchers are also working on ideas for using these technologies to provide warnings to people with allergies when they pick up certain food, for food safety, for sorting waste, and much more.

Cars Computer-controlled brakes, movies, and games for passengers, and the use of GPS navigation are examples of pervasive computing technologies that are already widespread today. Solutions offering automatic alarms in

places where the road is slippery by means of communication with other cars, and other traffic safety initiatives are also underway.

IT In this sector pervasive computing offers new solutions such as public printers that can be used via mobile phones, and the automatic storing of personal data such as digital photos. The use of electronic chips in passports is another example.

Military Last, there are a number of military applications such as unmanned planes, tanks, operating rooms, weapons that can only be fired by the rightful owner, etc.

All these applications are based on units with built-in computers and communication capabilities, of which we have probably only seen the tip of the iceberg. Pervasive computing typically involves units with limited resources in terms of computational power, memory, bandwidth, and battery, but this does not necessarily have to be true.

Pervasive computing covers both current and future technologies, so in order to separate issues today from the possibly different issues of tomorrow, we have divided different use of the technology into three different categories. These definitions are inspired by Forrester's definitions of **The Executable Internet**: *Intelligent applications that execute code near the user to create rich, engaging conversation via the net* and **The Extended Internet**: *Internet devices and applications that sense, analyze, and control the real world.* Our definitions, as they appear in our report to the former Danish National Council of IT security [160], are the following (in order of increasing technological complexity):

ID-in-everything. This level is already used commercially, and is characterized by the use of passive units, by which we mean that the units can only report their identity, and possibly raw sensor input. All calculations that do not have anything to do with the reporting of the identity are placed in the infrastructure that the unit is communicating with. The most prominent member of this category is the RFID tag.

RFID tags are small wireless devices that react to an electromagnetic field generated by an RFID reader. When they enter the electromagnetic field it induces an electric current in the tag, which can then emit some prestored information or perform some computation. The computing power one can assume an RFID tag to have, however, is severely limited in many applications by requirements for extremely low cost tags. Today, RFID is primarily used in the supply chain. By putting an RFID tag on a product, or, for example, a pallet with products, one can easily monitor and control the logistics from the manufacturer to the shop. However, as useful as this is to these parties, the possibility of uniquely identifying objects that will eventually be sold to customers, has raised a lot of privacy concerns.

Besides the supply chain RFID can be used in many other places. In the amusement park Legoland, kids can be given a wristband with an RFID tags, which parents can use to track the whereabouts of their child, and in most ski resorts, guest are given an RFID tag which gives them access to the ski lift.

The Mexican legal system offers a more exotic example, where employees in the Mexican Public Prosecutor's office have had an RFID chip implanted under the skin to identify them when accessing legal documents. A similar technology is used in some night clubs where certain VIP guests are identified by an RFID tag under their skin, which they use to gain fast access to the club, as well as to pay for their drinks.

So far we have only mentioned RFID tags, since they are the prime representative of this category. However, many other technologies share a common property with RFID tags, such as mobile phones and laptop computers: They are uniquely identifiable and tie a physical object to a virtual identity.

Services-in-everything. This category is what Forrester call the *The Executable Internet*, and is characterized by the use of more active units that can perform calculations and influence their surroundings. These units are not just identifiable, they are also fitted with sensors, actuators, and applications whereby they are able to offer various forms of services. Basically, we can say that in this category, technology assist the user in his actions. This technology is already partly realized, for example via PDAs, digital cameras and other things that can communicate, and that are thereby on the Internet in one form or another. They are not limited to just reporting an identity, but can connect to other units, and back-end systems, to offer more complex services.

Agents-in-everything. This category is called *The Extended Internet* by Forrester, and is characterized by units that are not only active, but also autonomous, i.e. they send data and influence their surroundings on their own initiative. Examples of this could be software agents that look for your favorite wine, and automatically order and pay for it (after having asked the bottle directly about its temperature throughout its lifetime), or it could be a software agent who informs another person about the nearest pharmacy where he can get that specific kind of medicine he needs, which your agent knows since you use the same kind of medicine.

The difference from the services-in-everything category, lies in the scope, and in the fundamental difference that a great deal of the responsibility for decisions is handed over to technology. The transfer of responsibility consists of a great number of decisions being made by so-called agents. This includes simple tasks such as the answering of telephone calls and calendar functions, but also financial transactions and concerns about supplying personal data are handled automatically. Another fundamental difference from the other two categories, are that we can no longer restrict ourselves to communicating with units that are connected to a local system. We will have to deal with foreign systems and devices all the time.

Technologies in this category are not yet available, but scenarios describing the expected use of such technologies, have been written by a think tank under the European Union [87].

1.2 Privacy

In this section, we establish some concepts, which can be used to relate the systems and solutions we later propose, to the real world. Solutions to privacy problems are not discussed in this section. We will also talk about how privacy problems manifest themselves in pervasive computing, but before we can do that, we take a brief look at what privacy is, why people care about it, and why we should be concerned about protecting it.

The Merriam-Webster dictionary defines privacy as *the quality or state of being apart from company or observation* and *freedom from unauthorized intrusion*. According to Altman [2], privacy is the result of different processes that continuously work in an area around each individual, which he refers to as the self-environment boundary. More specifically, Altman defines privacy as a *self-environment boundary regulation process based on dynamic, social, dialectic normalization of desired privacy to attained levels*. But which processes and why? Gavinson uses the term *control* to describe privacy and divides privacy into three basic elements as seen below [97]. Boyle claims that we regulate the self-environment boundary using exactly these three genres of control [44].

Solitude Control over ones interpersonal interactions with other people. This is something we as humans often handle automatically, by regulating which stimuli are sensed from our surroundings.

Confidentiality Control over other peoples access to ones information. More specifically it is about the fidelity of which information is accessed by others. Fidelity is the subjective perceived understanding of the correctness and detail of the information captured, so it is more about the level of details you give away, than it is about giving it all away or keeping it all secret.

Autonomy Autonomy is not control over our behavior, since our behavior is part of what defines us as individuals and hence some parts of it, is outside of our control. Instead autonomy is control over the observable manifestation of our identity. It combines factors such as behavior, appearance, impression, and self-definition. It could also be referred to as *freedom of will*

These controls are primarily exerted through individual and social human behavior, but could for example also be exerted through law or technology. They work concurrently and can have a strengthening or weakening implication on each other – for example, being able to do whatever you like can influence the solitude of other people. Choosing to spend a lot of time with other people will likely increase the amount of information you give away about yourself and hence decrease the confidentiality of your personal information. The keyword here is choice. The self-environment boundary is regulated through control, but without choice we have no control. In order to have control over our confidentiality we must be able to disclose information as well as keeping it secret.

In order to control solitude we must have the choice between staying alone or interacting with other people.

But why is privacy even important? Why don't we just say that if we have nothing to hide, we don't care what other people know about us. That topic in itself is worthy of a lengthy report, and several books and articles have been written about it (e.g. [170]). Here we will just sketch a very brief overview.

When people use the argument *if you are not doing anything wrong, you have nothing to hide* they are thinking of privacy only as confidentiality of some observable facts about a person, but that is missing the point, since privacy is much more than that. The most common answers one might come up with in response to the question above could be *if I'm not doing anything wrong, then you have no reason to watch me* or *because you might do something wrong with my information.* While these answers are certainly valid (especially the latter), and while there is certainly good reason to watch the watchers, they miss an important point in that they focus only on privacy as keeping personal information private. Rössler states in her book *The Value of Privacy* that *No matter if you do something wrong or illegal – just the fact that you feel watched or know you are being watched has an impact on the way you behave* [170]. Going back to the genres of control, being forced in this way to alter your behavior, the way you interact with other people, etc. restricts your control over your solitude and autonomy, and being watched takes away your control over personal information about you, which affects your confidentiality. Altman says the following about privacy: *At the psychological level, privacy supports social interaction which, in turn, provides feedback on our competence to deal with the world which, in turn, affects our self-definition.* In other words, privacy affects who we are and how we define ourselves. That explains why many people do not like being under surveillance, even though they cannot explain exactly why.

The above does not say much about how privacy violations occur, and thus does not help us design technologies to protect privacy. We will touch this area in the next section, but for a more detailed classification, we refer to the model by Lederer, Mankoff and Dey [135]. This model provides some tools for analyzing conditions for privacy implications of a given phenomenon, by looking at topics such as how does a privacy violation occur, what is disclosed and why is it disclosed.

1.2.1 Related to Pervasive Computing

We now get to the issue of how privacy violations occur in some pervasive computing applications, by starting with the most prominent technology in the ID-in-everything category, namely the RFID tag. Systems using RFID tags use them to map physical objects to virtual identities, by assigning each object an identity code, and a database then keeps track of which identity codes belong to which objects. Since RFID tags only have a very limited range (a few meters at most) when an RFID reader sees an identity code, it can conclude that the object with that code is close to the reader. This can then be used for providing access control, looking up the price of some goods, turning on the light, giving the washing machine instructions on how to wash that particular

piece of clothing that the tag is attached to, or whatever one wants to do in response to the proximity of a specific physical object. In addition to a database with the mapping of identity codes to physical objects, it is of course also possible to build up a database of the observations registered about a specific code. On the basis of this, there are three types of threats to privacy in RFID scenarios: Reading of identity codes, the misuse of existing databases with registered codes, and finally a combination of the two.

As identity codes are sent over the air, it is easy to intercept them without being noticed. This threat is basically more serious the greater the distance from which identities can be intercepted.

The ability to read an identity code from an RFID tag, discloses nothing about the physical object associated with this identifier, unless you have access to the database that associates the code with the object and/or user (it still poses some security risks if the tag is used for authentication). However, it does allow tracking of an anonymous user's movements around in the public space, such as a shopping center or on the street. Note that the possibility of carrying out surveillance is neither new nor particular to pervasive computing. It has been possible for many years with video surveillance for example, but with ID-in-everything it will be very easy and inexpensive to automate the collection and processing of these data. Of course such tracking of unknown identity codes is only useful if a database is kept up to date with codes and the context in which they are read.

This leads us on to the next threat, which is the misuse of databases containing identity codes that can be traced to a particular person, by linking different pieces of information together. Such analysis is already commonplace for detecting credit card fraud, and identifying consumption patterns, and in the same way it is easy to imagine the value of analysis of consumption patterns on the basis of information from the use of RFID in the retail sector. In connection with misuse of databases, it is important to understand that an attacker must have access to the database, and will therefore often be an insider. The threat to privacy in connection with tracking therefore comes, to a great extent, from the organizations and their employees who control these databases. In comparison with the more direct and tangible threat of privacy by reading what people on the street have in their shopping bags, this threat is less obvious, because it is based on an analysis of a quantity of data that has been collected, where each individual case of scanning can appear innocent.

The last threat is the combination of scanning of identity codes and the (mis)use of databases. A product with an RFID tag bought in one shop will also be able to be read in the shopping bag by another shop that also uses RFID. The difference here is that as well as being able to follow a certain identity code, there is also access to the database that links this code to more interesting data. Depending on what data material is available, the second shop will have an idea of what products the consumer might be interested in, and possibly who we are, or even what we have previously bought in various shops. This type of attack on privacy will generally be more complex than attacks that merely analyze a single database, both because they require access to the relevant databases and because the attack most likely will involve several organizations that are

working together.

Everything said so far in this section is related to RFID tags, but many other technologies pose similar privacy threats. The hardware address of the wireless network card in laptops and PDAs, Bluetooth, GSM, etc. are all examples of technologies with a unique identity code that can be read over the air and (ab)used just as the code sent by an RFID tag. The key element in the ID-in-everything category is that units can be identified, i.e. a depiction of physical objects – whether it will be objects or people, or objects that represent or that are associated with people, such as a train ticket – in the virtual space. It is however worth pointing out that in all cases except for RFID, the unique identifier is usually a byproduct, meaning that the technology doesn't exist with the purpose of broadcasting a unique identifier, but it does so because of the inner workings of the technology. For example the Bluetooth ID is necessary for addressing devices, but can also be used to uniquely identify a device without the knowledge of the device owner. A GSM telephone transmits a unique IMEI number that can be read at a distance of several kilometers, a laptop computer that is set up for wireless communication sends out a unique address, etc.

As with ID-in-everything, the identification of units is a very important element in services-in-everything as well. However, units do not just have to send an identity code to a reader, they are able to perform computation and thus might leak more information than just their identity. Take for example a credit card. When using it to pay for some goods it not only gives the store the credit card number, but ties your identity to the card, and hence the transaction. The store can keep records of what you bought and how much money you spent, the bank knows where and when you shopped and how much you spent there, etc. Another example is systems used for authentication, where the purpose is just to prove that the user is allowed to access a resource. However, the protocol used for this, might leak information that allows a third party to uniquely identity that particular user. The good news is that if the system is designed properly, we have a lot more control in this category. We can design protocols that do not leak personal information, we can keep personal data on the device instead of sending it off to third parties, etc.

Compared to the previous two categories in the agents-in-everything category, we have intelligent agents interacting with many foreign systems they have never interacted with before, and with whom the user is not at all familiar. Furthermore, the agents will often decide on behalf of the user which personal information to disclose. Since scenarios in this area are still futuristic, we are not going to spend time on how to deal with privacy issues here, as it is still unclear how exactly things are going to work. We note however, that unless people designing these systems in the future are very careful, there will be a whole new wave of privacy violations.

1.3 Authentication

A credential is an attestation of qualification, competence, or authority, issued to some subject. Credentials can be anything from simple identifiers such as

your name, birthday, biological traits, etc., to statements such as being allowed to access a specific resource. However, before a credential can be accepted, it must be verified. Stating a claim such as your name is known as identification, whereas verification of that claim is known as authentication. In this section we introduce two fundamentally different techniques for authentication, and briefly survey some of the concepts we will return to later in this thesis.

Symmetric key authentication. Probably the most well known form of authentication to most people, is the username/password combination. Users log on to systems by first identifying themselves with a username, and then authenticate themselves by typing in the password associated with that username. The system then checks if the password matches the one stored for that user (preferably hashed in some way, e.g. by using PKCS#5 [126]), and if it does, then the user is granted access. At first sight this seems fine, but there are a number of problems with this approach. First of all, if authentication is done over a network, everyone monitoring the traffic can see the password. Second, the user reveals his secret to the system, but since the system is not authenticated towards the user, he has no way of knowing if he is indeed talking to the system he thinks he is, or if he is revealing his password to a malicious system. A better method would be to execute a zero-knowledge proof between the user and the server, demonstrating that the user does indeed have the right password, without revealing it to anyone [104]. Often we are interested in more than just authenticating the user's identity. We also want to exchange some cryptographic key material, which can later be used to protect the communication channel between the user and the server. Many protocols have solved this problem, with varying security guarantees. The most important property is of course that the password stays private, but also that monitoring the communication channel between the user and the server, does not allow an adversary to mount an off-line brute force attack, trying to guess the password. The only way to try to guess the right password, should be to query the server once for each guess, since the server can then quickly determine if an attack is in progress, and deny further attempts. The latest example of such a protocol is by Canetti, Halevi, Katz, Lindell, and MacKenzie [62] which is provably secure under Canetti's Universal Composability framework. Then of course there is the whole issue whether or not passwords are secure enough to be used for authentication. Many users have a tendency to pick trivial and easy to guess passwords.

Many authentications do not take place between a user and a system, but between entities in a system itself, so let us take another example; RFID tags. The first problem is that standard RFID tags do not perform any kind of authentication, only identification. They send out their credential, which is just an identity code, but it is not possible to authenticate the tag. The identity code could have been seen by anyone, and now being retransmitted. That causes two problems. As we have discussed, sending out a unique identity code is bad from a privacy point of view, but it also makes standard RFID tags unsuitable for authenticating people or objects they are attached to. An obvious solution

would be to equip the tag with a symmetric key shared with the reader, and then use an authentication protocol between the tag and the reader, similar to what we have just seen between a user and a system. However, one must know exactly what the purpose of the authentication is. Assuming that the authentication protocol is secure, the only thing authenticated would be, that the tag is out there somewhere. It does not have to mean that the tag is close to the reader, since there could be a device close to the reader, communicating with a device close to the tag, relaying messages between the tag and the reader. Such an attack is called the proxy attack, or the mafia fraud attack, and can be quite serious when systems rely on transmission range to authenticate the presence of some device, e.g. for controlling access to a building. The only known way to protect against such an attack, is to use so-called distance bounding protocols that measures the time it takes for the tag to respond, and can detent if the response is coming from a device close to the reader, or from far away [47].

From a privacy point of view there are also some serious issues with the protocols we have described so far. In the first example, the user will always send his username before the protocol starts, and in the RFID case, the tag just sends out an identity code to any reader nearby.

Using public key cryptography this problem is easily solved. Simply let the tag encrypt its identity under the public key of the reader, before broadcasting it[1]. The reader could then decrypt the message, and learn the identity of the tag. However, as we have mentioned, RFID tags are small devices incapable of performing public key cryptography. This privacy problem started a fairly intensive research in private authentication protocols, based on symmetric key techniques. One of the first proposed solutions for RFID tags were the so-called hash locks by Weis, Sarma, Rivest, and Engels [190], but many followed [14, 15, 124, 146, 153, 154, 185]. However, it was unclear what kind of adversaries these protocols were supposed to protect against. In 2005 Avoine classified a range of adversarial capabilities [13] and soon after Juels and Weis described a single, but very powerful, adversary, with a simple definition called *strong privacy* [125]. Burmester, Le, and Medeiros [51] have also proposed a security definition based on Canetti's Universal Composability framework. We will return to the issue of private RFID authentication in Chapter 3, but note that these techniques are not only for RFID tags, but can be used in general for private authentication based on symmetric keys.

Public key authentication. The use of symmetric keys is somewhat limited, due to the fact that the both parties must agree on cryptographic keys in advance. Except for the problems of distributing the keys, this also does not permit properties such as non-repudiation. However, if we turn to authentication based on public key techniques, we can solve many other interesting security problems related to authentication.

These techniques are centered around digital signatures. A digital signature scheme can be used to sign messages, such as documents or login requests, or a

[1]This of course assumes that it is not possible to determine if two ciphertexts are the encryption of the same value

trusted third party can issue credentials, by signing certain statements about an entity. Furthermore, digital signature schemes are often used as building blocks in more complex protocols. The most well-known (and used) digital signature schemes today are RSA and DSA. In Chapter 2 we describe these definitions in more details.

The downside is that uses of digital signatures as the digital equivalent of a pen and paper signature have privacy issues, since they rely on unique identifiers, namely the verification key, or the credentials which have been signed by a third party. Luckily this does not have to be the case. There are several ways to achieve some kind of private authentication, but the two most important techniques for achieving authentication, while still preserving anonymity, are group signatures and anonymous digital credentials. We are going to review these techniques in Chapter 4, but we will briefly mention them here for completeness.

Group signature schemes, first introduced by Chaum and van Heyst [71], allow a member of a group to sign a message on behalf of the group in a way so that the verifier cannot determine which group member signed the message, only that it was indeed signed by a member of the group. Among other things, this enables us to construct credentials, that show nothing more than the fact that the user has been (together with many others) authorized to access a given resource. While group signatures are limited to signing on behalf of a single group, anonymous digital credentials are the privacy friendly version of having a X.509 certificate containing all your personal information, signed by a trusted third party. As in the case of a single X.509 certificate, the user will have a number of credentials certified by a trusted third party, but the difference is that the user decides which credentials he wants to show possession of, instead of just sending the entire certificate to the verifier. This is done by proving in zero knowledge, certain properties of his credentials. Anonymous digital credentials were first introduced by Chaum in 1985 [68].

1.4 Outline

First we examine the issue of privacy and authentication related to RFID systems in Chapter 3. We look at how privacy can be protected, and give an overview of some related work in that area. Then we present our results about tradeoffs between security and efficiency in RFID systems. More specifically we look at the definition of *strong privacy* by Juels and Weis [125], why efficient private RFID authentication protocols are not possible in this (strong) model, and then we propose more realistic assumptions about the adversarial behavior, that allows us to design more efficient protocols that still remain secure in many practical applications of RFID.

Next, we look past RFID and take a more general look at anonymous authentication. In Chapter 4 we survey some cryptographic techniques to protect privacy, and then we will look at our definition and design of what we call an unclonable identification scheme.

In Chapter 5 we turn to other issues regarding authentication in pervasive

computing, where privacy is not a primary concern. We present two of our results in this area. First we briefly review our proposal for a login system for a hospital environment where usability was the key factor, and then we look at an anti-theft policy for pervasive computing devices where the key property is to authenticate the user towards a system of devices in his own home. We describe how the policy can be implemented, and as part of that, we also propose two different protocols for performing key exchange between devices in a user friendly manner. The aim of this is to highlight the relation between security and usability, and propose some solutions where these two are not mutually exclusive.

Then in Chapter 6 we examine batch verification of signatures. Short signatures are important in many applications, especially in the pervasive computing realm, but unfortunately most short signatures are slow to verify. First we survey related work on batch verification, and conclude that no suitable batch verifiers exist for short signature schemes. We propose the first batch verifier for messages from many signers without random oracles and with a verification time where the dominant operation is independent of the number of signatures to verify. We further propose a new signature scheme with very short signatures, for which batch verification for many signers is also highly efficient. In the last part of the chapter, we provide a general framework for designing batch verifiers, and we apply this framework to batch verification of certain types of anonymous authentication such as group and ring signatures. Furthermore, we implement some of the schemes and provide an in depth performance analysis, which highlights a few issues we think implementers need to be aware of.

Chapter 2

Preliminaries

In this chapter, we introduce notation and basic concepts used in the rest of this thesis.

2.1 Pairings

Since many of the schemes and techniques discussed in this thesis are pairing-based, we briefly review pairings and their relevant properties.

Let PSetup be an algorithm that, on input the security parameter 1^τ, outputs the parameters for a bilinear pairing as $(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e})$, where $\mathbb{G}_1 = \langle g_1 \rangle, \mathbb{G}_2 = \langle g_2 \rangle$ and $\mathbb{G}_T$ are groups of prime order $q \in \Theta(2^\tau)$. The efficient mapping $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is both: (*bilinear*) for all $g \in \mathbb{G}_1$, $h \in \mathbb{G}_2$ and $a, b \leftarrow \mathbb{Z}_q$, $\mathbf{e}(g^a, h^b) = \mathbf{e}(g, h)^{ab}$; and (*non-degenerate*) if g generates $\mathbb{G}_1$ and h generates $\mathbb{G}_2$, then $\mathbf{e}(g, h) \neq 1$. This is the general case, called the *asymmetric* setting. A specialized case is the *symmetric* setting where $\mathbb{G}_1 = \mathbb{G}_2$. We will always write group elements in the multiplicative notation, although the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are actually implemented as additive groups.

In case we are only interested in the symmetric case, we will write that PSetup is an algorithm that, on input the security parameter 1^τ, outputs the parameters for a bilinear pairing as $(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$, where $\mathbb{G} = \langle g \rangle$ and $\mathbb{G}_T$ are of prime order $q \in \Theta(2^\tau)$. This is similar to the definition for the asymmetric case.

Since we are looking at various schemes based on these groups, we are interested in groups that have the smallest representation of the group elements, we want to know if schemes in the symmetric setting can be moved over to the asymmetric setting, and finally want to be able to verify whether a given element is a member of a specific group.

Size of Group Elements. Pairings are constructed such that $\mathbb{G}_1$ and $\mathbb{G}_2$ are groups of points on some elliptic curve E, and $\mathbb{G}_T$ is a subgroup of a multiplicative group over a related finite field. All groups have order q. The group of points on E defined over $\mathbb{F}_p$ is written as $E(\mathbb{F}_p)$. Usually it is the case that $\mathbb{G}_1$ is a subgroup of $E(\mathbb{F}_p)$, $\mathbb{G}_2$ is a subgroup of $E(\mathbb{F}_{p^k})$ where k is the embedding

degree, and $\mathbb{G}_T$ is a subgroup of $\mathbb{F}^*_{p^k}$. In the symmetric case $\mathbb{G}_1 = \mathbb{G}_2$ is usually a subgroup of $E(\mathbb{F}_p)$.

In the following, we use numbers for security comparable to 1024 bit RSA. The MOV attack by Menezes, Vanstone, and Okamoto states that solving the discrete logarithm problem on a curve reduces to solving it over the corresponding finite field [143]. Hence the size of p^k must be comparable to that of a RSA modulus to provide the same level of security, so elements of $\mathbb{F}_{p^k}$ must be of size 1024. But the size of the finite field is not the only thing that matters for security. The group order q must also be large enough to resist the Pollard-ρ attack on discrete logarithms, which means that $q \geq 160$. Now assume that $|p| = |q| = 160$, then we would need an embedding degree $k = 6$ to get the size of the corresponding field close to the required 1024 bits. However, we could also let $|q| = 160$, $|p| = 512$, and choose $k = 2$ to achieve the same. Both these options have their advantages and disadvantages as discussed by Koblitz and Menezes [130].

We have talked about how many bits are required to represent elements in the finite fields, but what about the groups $\mathbb{G}_1$ and $\mathbb{G}_2$? Since they are subgroups of a curve over the field, they are represented by their coordinates (x, y) which are elements of the field, and hence one would expect their size to be twice the size of an element in the field. However one only needs to represent x and the sign bit of y in order to recompute y later. Also, in some cases (when $\mathbb{G}_2$ is the trace zero subgroup) elements of $\mathbb{G}_2$ can be represented as elements of the field $E(\mathbb{F}_{p^{k/2}})$ instead, which only requires half the space [130].

To summarize: In the asymmetric setting, the best we can hope for are group elements in $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ of size 160, 512 and 1024 bits respectively. In the symmetric setting it seems the best curve is a supersingular curve with $k = 2$, which means that elements of $\mathbb{G}_1 = \mathbb{G}_2$ and $\mathbb{G}_T$ will be of size 512 and 1024 bits respectively. Finally, an important thing to keep in mind is that no matter the order of the groups, performance is dominated by the operations in the underlying finite field.

So how should one choose a curve when implementing a scheme? Unfortunately there is no simple answer. It depends on what group is used the most in a given scheme, if size of group elements or efficiency is most important, if the scheme requires certain properties that are only available on some curves (such as a homomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$), etc. Galbraith, Paterson, and Smart have a more detailed discussion about these issues [96].

Testing Membership. Some schemes require that the input lies in a specific group, but since the input might be supplied by the adversary we cannot trust this to hold. We have to check it. For example, the proofs in Chapter 6 require that elements of purported signatures are members of $\mathbb{G}_1$ and *not* $E(\mathbb{F}_p) \setminus \mathbb{G}_1$, but how efficiently can this fact be verified? Determining whether some data represents a point on a curve is easy. The question is whether it is in the correct subgroup. Determining whether some data represents a point on a curve is easy. The question is whether it is in the correct subgroup. Assume we have a bilinear map $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. In the schemes we use, the input is from $\mathbb{G}_1$, so this is

the group we are interested in testing membership of.

In most cases, if we want to check if an element y lies in $\mathbb{G}_1$, we must check if $y^q = 1$. While this might seem inefficient, it is usually not a problem when working with pairing based schemes, since the time required for a single exponentiation is insignificant compared to the time required for computing a pairing. More about this in Chapter 6. Chen, Cheng, and Smart [72] discuss membership testing for bilinear groups in more details.

2.2 Complexity Assumptions

In the following chapters, we will refer to these complexity assumptions.

Assumption 2.1 (Computational Diffie-Hellman [84]) *Let g generate a group $\mathbb{G}$ of prime order $q \in \Theta(2^\tau)$. For all p.p.t. adversaries $\mathcal{A}$, the following probability is negligible in τ:*

$$\Pr[a, b, \leftarrow \mathbb{Z}_q; z \leftarrow \mathcal{A}(g, g^a, g^b) : z = g^{ab}].$$

Assumption 2.2 (Decisional Bilinear Diffie-Hellman [34])
Let $\mathsf{PSetup}(1^\tau) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$, where g generates $\mathbb{G}$. For all p.p.t. adversaries $\mathcal{A}$, the following probability is at most 1/2 plus a negligible function in τ:

$$\Pr[a, b, c, d \leftarrow \mathbb{Z}_q;\ x_0 \leftarrow \mathbf{e}(g,g)^{abc};\ x_1 \leftarrow \mathbf{e}(g,g)^d;\ z \leftarrow \{0,1\};\\ z' \leftarrow \mathcal{A}(g, g^a, g^b, g^c, x_z) : z = z'].$$

Assumption 2.3 (LRSW [141]) *Let $\mathsf{PSetup}(1^\tau) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$, $X, Y \in \mathbb{G}$, $X = g^x$, and $Y = g^y$. Let $\mathcal{O}_{X,Y}(\cdot)$ be an oracle that, on input a value $m \in \mathbb{Z}_q^*$, outputs a triple $A = (a, a^y, a^{x+mxy})$ for a randomly chosen $a \in \mathbb{G}$. For all p.p.t. adversaries $\mathcal{A}^{(\cdot)}$, the following probability is negligible in τ:*

$$\Pr[(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e}) \leftarrow \mathsf{PSetup}(1^\tau); x \leftarrow \mathbb{Z}_q; y \leftarrow \mathbb{Z}_q; X = g^x; Y = g^y;\\ (m, a, b, c) \leftarrow \mathcal{A}^{\mathcal{O}_{X,Y}}(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e}, X, Y)\ :\ m \notin Q\ \wedge\ m \in \mathbb{Z}_q^*\ \wedge\\ a \in \mathbb{G} \wedge\ b = a^y\ \wedge\ c = a^{x+mxy}]$$

where Q is the set of queries that $\mathcal{A}$ made to $\mathcal{O}_{X,Y}(\cdot)$.

2.3 Zero-Knowledge Proofs and Proofs of Knowledge

Zero-knowledge proofs were introduced by Goldwasser, Micali, and Rackoff in 1985 [105] and constitute a central class of cryptographic protocols. In an interactive proof system there are two players, namely the prover $\mathcal{P}$ who wants to convince the verifier $\mathcal{V}$ about the validity of some statement. There are two requirements for interactive proof systems. *Completeness* means that if the statement is true then $\mathcal{P}$ should be able to convince $\mathcal{V}$ of this, and *soundness* means that if the statement is false, no prover should be able to convince the verifier about this with probability greater than some function of the input length. More formally:

Definition 2.1 (Interactive Proof System) *Interactive probabilistic turing machines $\mathcal{P}(\cdot), \mathcal{V}(\cdot)$ constitute an interactive proof system for a language L if:*

Completeness *If $x \in L$ then*

$$\Pr[P(x) \leftrightarrow \mathcal{V}(x) \rightarrow 1] = 1$$

Soundness *Let ν be a negligible function. For all $x \notin L$, for all provers $\mathcal{P}'$*

$$\Pr[\mathcal{P}' \leftrightarrow \mathcal{V}(x) \rightarrow 1] \leq \nu(|x|)$$

An interactive proof system is usually only of interest if the verifier cannot hilself determine whether $x \in L$ or not, which is the reason the verifier is limited to probabilistic polynomial time. However, in practise one might ask why $\mathcal{P}$ can have more computational power than $\mathcal{V}$. A variant of an interactive proof system, where $\mathcal{P}$ is also bounded by probabilistic polynomial time, and where his power comes from some additional auxiliary input, is called an *interactive argument*.

Definition 2.2 (Interactive Argument) *A pair of interactive probabilistic turing machines $\mathcal{P}(\cdot,\cdot), \mathcal{V}(\cdot)$ constitute an interactive argument for a language L if:*

Completeness *If $x \in L$ then there exist a witness ω such that*

$$\Pr[\mathcal{P}(x,\omega) \leftrightarrow \mathcal{V}(x) \rightarrow 1] = 1$$

Soundness *For all $x \notin L$, for all probabilistic polynomial time provers $\mathcal{P}'$*

$$\Pr[\mathcal{P}' \leftrightarrow \mathcal{V}(x) \rightarrow 1] \leq \nu(|x|)$$

Of course any language in NP has a trivial interactive argument. The prover can simply reveal ω to $\mathcal{V}$, but the interesting thing about interactive proofs and arguments, is that it is possible to convince $\mathcal{V}$ of this without revealing any information about ω. An interative proof system and interactive argument with this property is called a *zero-knowledge proof system* and *zero-knowledge argument*, respectively.

Informally, the definition of a zero-knowledge proof system (respectively argument) is that we can replace $\mathcal{P}$ with a simulater, that without talking to $\mathcal{P}$ at all, just assumes that $x \in L$ and computes whatever is needed to make $\mathcal{V}$ accept without $\mathcal{V}$ noticing that he is not talking to the prover. Soundness is still preserved, since the simulator has some power than $\mathcal{P}$ does not. Namely, if it should issue a message that would cause the verfier to reject, it can simply rewind and issue a different message. More formally we first need to define a simulator:

Definition 2.3 (Simulator) *A probabilistic polynomial time turing machine $\mathcal{S}$ is called a simulator for machine $\mathcal{A}$'s interaction with machine $\mathcal{B}$ on input x, if for all inputs y polynomial in $|x|$ the following holds:*

$$VIEW_{\mathcal{A}}(\mathcal{A}(y,x) \leftrightarrow B_x) \sim VIEW_{\mathcal{A}}(\mathcal{A}(y,x) \leftrightarrow \mathcal{S}(x))$$

We can now formally define a zero-knowledge proof system.

Definition 2.4 (Zero-Knowledge Proof System) *A proof system is called a zero-knowledge proof system for a language L if for all verifiers $\mathcal{V}^*$ there exist a simulator $\mathcal{S}$ for all $x \in L$, for $\mathcal{V}^*$'s interaction with $\mathcal{P}(x)$ on input x.*

For completeness we also define a zero-knowledge argument:

Definition 2.5 (Zero-Knowledge Argument) *A zero-knowledge argument for a language L is an argument where the following holds: For all verifiers $\mathcal{V}^*$ there exist a simulator $\mathcal{S}$ for all $x \in L$, for $\mathcal{V}^*$'s interaction with $\mathcal{P}(x, \omega)$ on input x.*

Zero-knowledge can be either *perfect*, *statistical* or *computational* depending on the simulator. Perfect means that the two views in the simulation are identical, statistical means that there is a negligible difference between the real and the simulated view, and computational means that an observer restricted to polynomial time tests cannot distinguish the real and the simulated view except with negligible probability.

Proofs of Knowledge. A proof of knowledge protocol is a protocol that allows a prover, $\mathcal{P}$, to convince a verifier, $\mathcal{V}$, that he knows a certain quantity ω that satiesfies some polynomial time computable relation R. We view this as R taking two inputs. One that is known to both parties and one that is known only to $\mathcal{P}$. Proofs of knowledge was first introduced by Feige, Fiat, and Shamir [90] and Tompa and Woll [183], and further redefined by Bellare and Goldreich [24].

Clearly revealing ω is a proof of knowledge, but it is more interesting that it is possible to prove possesion of ω without revealing any information about it. A protocol satisfying this condition is called a *zero-knowledge proof of knowledge* protocol. Informally, there exist a knowledge extractor $\mathcal{K}$ that uses $\mathcal{P}$ and $\mathcal{V}$ to compute ω such that $R(x, \omega)$ is satisfied, where x is the common input. The intuition behind this definition is that since it is possible to extract ω, it must have been known to $\mathcal{P}$, and hence the prover cannot make $\mathcal{V}$ accept an invalid proof. More formally:

Definition 2.6 (Proof of Knowledge [140])
Let $R(\cdot, \cdot)$ be a polynomially computable relation. Let $\nu(x)$ be such that $|\omega| \leq \nu(x)$ for all ω such that $R(x, \omega)$ holds, and assume that some such $\nu(x) = poly(|x|)$ is efficiently computable. A verifier $\mathcal{V}$ is a knowledge verifier with respect to R if the following conditions hold:

Non-triviality *There exist a prover $\mathcal{P}$, such that for all x, ω, if $R(x, \omega)$ holds, then*

$$\Pr[\mathcal{P}(x, \omega) \leftrightarrow \mathcal{V}(x) \rightarrow b : b = 1] = 1$$

Extraction (with error $2^{-\nu(x)}$) *There exist an extractor algorithm $\mathcal{K}$ and a constant c such that for all x, for all adversaries $\mathcal{A}$, if*

$$p(x) = \Pr[\mathcal{A} \leftrightarrow \mathcal{V}(x) \rightarrow b : b = 1] > 2^{-\nu(x)}$$

then, on input x and with access to $\mathcal{P}$, $\mathcal{K}$ computes a value ω such that $R(x, \omega)$ holds, within an expected number of steps bounded by $\frac{(|x|+\nu(x))^c}{p(x)-2^{-\nu(x)}}$

Based on a commitment scheme and, for instance, one of the protocols for graph 3-colorability from [104], we can build generic proofs of knowledge for any binary relation R that can be checked in polynomial time. The protocol in its basic form is a three move protocol where the second message is a one-bit challenge from the verifier. When we work with security parameter τ, we may compose *sequentially* τ instances of this protocol, to obtain a zero-knowledge proof of knowledge for R with negligible soundness error.

We will be concerned with proofs of knowledge of the following form, where x is common input to $\mathcal{P}$ and $\mathcal{V}$, and ω such that $R(x, \omega)$ is satisfied, is private input to $\mathcal{P}$. Furthermore we assume that $\mathcal{P}$ and $\mathcal{V}$ are probabilistic polynomial time turing machines:

1. $\mathcal{P}$ sends a message a.
2. $\mathcal{V}$ responds with a random τ-bit string e.
3. $\mathcal{P}$ sends a reply z and $\mathcal{V}$ either accepts or rejects based on x, a, e, z.

Definition 2.7 (Σ-protocol) *A protocol is a Σ-protocol for a relation R if :*

Completeness *The protocol follows the three move form above, and if $\mathcal{P}$ and $\mathcal{V}$ follow the protocol, $\mathcal{V}$ will always accept.*

Special Soundness *From any common input x and accepting conversations (a, e, z) and (a, e', z') on input x where $e \neq e'$, one can efficiently compute ω such that $R(x, \omega)$ is satisfied.*

Special Honest Verifier Zero-knowledge *There exist a polynomial time simulator $\mathcal{S}$ which on input x and a random e, outputs an accepting conversation (a, e, z) with the same probability distribution as conversations between the honest $\mathcal{P}$ and $\mathcal{V}$ on input x.*

2.4 Commitment Schemes

A commitment scheme is a two-party protocol between a committer and a receiver, consisting of two stages: *commit* and *opening*. The committer takes a value as input to the commit stage, and reveals this value in the opening stage. A protocol is a secure commitment scheme if at the end of the commitment stage, the receiver has no information on the committed value, and that there exist only one value that the committer can reveal in the opening stage.

A commitment scheme consists of a setup algorithm Setup that sets up the public parameters for the scheme, as well as a deterministic commitment algorithm Commit. Setup takes as input the security parameter τ and outputs the public commitment key ck corresponding to some message space $\mathcal{M}$. Commit takes as input ck, $m \in \mathcal{M}$, and a random bitstring r of length τ. To commit to a value $m \in \mathcal{M}$ the comitter generates a random r, computes $C = \mathsf{Commit}_{ck}(m, r)$ and sends C to the receiver. In the opening stage, the committer sends (m, r) to the receiver, who verifies $C = \mathsf{Commit}_{ck}(m, r)$. More formally, we define a commitment scheme as follows:

Definition 2.8 (Commitment Scheme)
A non interactive commitment scheme for a message space $\mathcal{M}$ consists of algorithms $(\mathsf{Setup}(\cdot), \mathsf{Commit}_{(\cdot)}(\cdot,\cdot))$ such that the following properties hold:

Computational Hiding *For all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function ν such that*

$$\Pr[ck \leftarrow \mathsf{Setup}(1^\tau); (m_0, m_1) \leftarrow \mathcal{A}(ck); r \leftarrow \{0,1\}^\tau; b \leftarrow \{0,1\}; \\ C_b \leftarrow \mathsf{Commit}_{ck}(m_b, r); b' \leftarrow \mathcal{A}(m_0, m_1, C_b) : \\ m_1, m_2 \in \mathcal{M} \wedge b' = b] \leq 1/2 + \nu(\tau)$$

Computational Binding *For all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function ν such that*

$$\Pr[ck \leftarrow \mathsf{Setup}(1^\tau); (m_1, r_1, m_2, r_2) \leftarrow \mathcal{A}(ck) : m_1, m_2 \in \mathcal{M} \wedge \\ m_1 \neq m_2 \wedge \mathsf{Commit}_{ck}(m_1, r_1) = \mathsf{Commit}_{ck}(m_2, r_2)] = \nu(\tau)$$

As stated, these definitions assume that the adversary is limited to probabilistic polynomial time. There also exist commitment schemes where the hiding (resp. binding) properties above, hold against a computationally unbounded adversary.

Unconditional Hiding For all unbounded adversaries $\mathcal{A}$ we require

$$\Pr[ck \leftarrow \mathsf{Setup}(1^\tau); (m_0, m_1) \leftarrow \mathcal{A}(ck); r \leftarrow \{0,1\}^\tau; b \leftarrow \{0,1\}; \\ C_b \leftarrow \mathsf{Commit}_{ck}(m_b, r); b' \leftarrow \mathcal{A}(m_0, m_1, C_b) : m_1, m_2 \in \mathcal{M} \wedge b' = b] \leq 1/2;$$

Unconditional Binding For all unbounded adversaries $\mathcal{A}$ we require

$$\Pr[ck \leftarrow \mathsf{Setup}(1^\tau); (m_1, r_1, m_2, r_2) \leftarrow \mathcal{A}(ck) : m_1, m_2 \in \mathcal{M} \wedge \\ m_1 \neq m_2 \wedge \mathsf{Commit}_{ck}(m_1, r_1) = \mathsf{Commit}_{ck}(m_2, r_2)] = 0$$

Unfortunately no commitment scheme can have both unconditional hiding and binding. Assume that such a scheme existed. If $\mathcal{P}$ sent a commitment $C = \mathsf{Commit}_{ck}(0, r)$ to $\mathcal{V}$, there must exist an s such that $C = \mathsf{Commit}_{ck}(1, s)$. If not, $\mathcal{V}$ could try all possible values of s and conclude that the commitment

could not have been to 1, thus violating the unconditional hiding property. But if such a value s existed, then $\mathcal{P}$ could find it, and claim that his commitment was to 1 instead of 0, thus violating unconditional binding.

A special flavor of commitment schemes are trapdoor commitment schemes that besides the properties above, satisty the following property as well

Trapdoor

- There exist a setup algorithm Setup' that on input the security parameter τ outputs a public key ck and a trapdoor t.
- If t, ck was generated by Setup' there exist an algorithm Alter which on input (t, C, m) outputs r such that $C = \mathsf{Commit}_{ck}(m, r)$.

Trapdoor commitment schemes are useful in cryptographic protocols that require simulation, since we can give the trapdoor t to the simulator, which makes it easier to create the view of the adversary.

2.5 Digital Signatures

Digital signature schemes play an important role in many areas. Not only as the digital equivalent of a pen and paper signature, but also for their use as building blocks to build more complex protocols. They were invented by Diffie and Hellman [84] and later formalized by Goldwasser, Micali, and Rivest [106], who defined a digital signature scheme as follows:

Definition 2.9 (Signature Scheme) *A signature scheme is a triple of probabilistic polynomial-time algorithms* $(\mathsf{Gen}(\cdot), \mathsf{Sign}_{(\cdot)}(\cdot), \mathsf{Verify}_{(\cdot)}(\cdot,\cdot))$*, where* Gen *is the key generation algorithm,* Sign *is the signature algorithm, and* Verify *is the verification algorithm, constitute a digital signature scheme for a family (indexed by the public key pk) of message spaces* $\mathcal{M}$ *if the following properties hold*

Correctness *If the message m is in the message space for a given public key pk, and sk is the corresponding secret key, then the output of* $\mathsf{Sign}_{sk}(m)$ *will always be accepted by the verification algorithm* Verify_{pk}*. More formally, for all values of m and τ:*

$$\Pr[(pk, sk) \leftarrow \mathsf{Gen}(1^\tau); \sigma \leftarrow \mathsf{Sign}_{sk}(m) : m \in \mathcal{M} \wedge \neg \mathsf{Verify}_{pk}(m, \sigma)] = 0$$

Security *Even if an adversary has oracle access to the signing algorithm which provides signatures on messages of the adversary's choice, the adversary cannot create a valid signature on a message not explicitly queried. More formally: Let Q be the list of messages the signing oracle* $\mathsf{Sign}_{(\cdot)}(\cdot)$ *has been queried on. For all probabilistic polynomial-time oracle turing machines* $\mathcal{A}^{(\cdot)}$ *with access to* $\mathsf{Sign}_{(\cdot)}(\cdot)$*, there exist a negligible function $\nu(k)$ such that*

$$\Pr[(pk, sk) \leftarrow \mathsf{Gen}(1^\tau); (m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}_{sk}(\cdot)}(1^\tau) : \mathsf{Verify}_{pk}(m, \sigma) = 1 \wedge m \notin Q)] = \nu(\tau)$$

A scheme secure under this definition is said to be *unforgeable*. An, Dodis, and Rabin [3] proposed the notion of *strong unforgeability*, where if $\mathcal{A}$ outputs a pair (m, σ) such that $\mathsf{Verify}(pk, m, \sigma) = 1$, then except with negligible probability at some point the signing oracle $\mathcal{O}_{sk}(\cdot)$ was queried on m and outputted signature σ exactly. In other words, an adversary cannot create a new signature even for a previously signed message.

Digital signatures as the equivalent of a pen and paper signature, have been used in practise for many years in form of X.509 certificates. However, the problem with this approach is that it is not privacy friendly. When verifying a signature you need to obtain the certificate (containing the public key of the user), which uniquely identifies that user, as well as reveals all his attributes stored in the certificate. This is especially a problem if the same certificate is used to authenticate the user towards different organisations.

However, some signature schemes have properties that can be used to build privacy friendly protocols. Some of these properties include the ability to obtain a signature on a message without revealing it and to prove that one possesses a valid signature on a committed value. Camenisch and Lysyanskaya noted that these properties are suffient for a signature scheme to be used in the construction of anonymous digital credentials [57, 140].

As an example of such a scheme, consider the signature scheme by Camenisch and Lysyanskaya [58], which is secure in the plain model under the LRSW assumption. Since we are going to use this scheme in future chapters, we describe it here.

Definition 2.10 (Camenisch and Lysyanskaya Signature Scheme) *Given a security parameter τ, let $\mathsf{PSetup}(1^\tau) \to (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$ output the parameters for a bilinear map.*

Gen *Choose $x \in \mathbb{Z}_p$ and $y \in \mathbb{Z}_p$ and set $X = g^x$ and $Y = g^y$. Then the secret key is $sk = (x, y)$ and the public key is $pk = (X, Y)$.*

Sign *On input message m, secret key sk and public key pk, choose a random $a \in \mathbb{G}$ and output the signature $\sigma = (a, a^y, a^{x+mxy})$.*

Verify *On input message m, public key pk and purported signature $\sigma = (a, b, c)$ check that the following equations hold*

$$\mathbf{e}(a, Y) = \mathbf{e}(g, b) \quad \textit{and} \quad \mathbf{e}(X, a) \cdot \mathbf{e}(X, b)^m = \mathbf{e}(g, c)$$

From now on we will refer to this scheme as the CL signature scheme. As mentioned before the CL scheme is interesting, because it allows us to obtain a signature on a message m without revealing m, and to prove in zero-knowledge that we know a signature on a message [57, 140]:

To obtain a signature on the message m without revealing m, we give g^m as input to the signer instead of m. The algorithm still works if we choose $\sigma = (g^r, a^y, a^x M^{rxy})$, because $a^x M^{rxy} = a^{x+mxy}$. Of course the user needs to prove knowledge of m in order for the signature scheme to remain secure. Note that while this hides the message m, the value g^m is still a unique identifier, if

the message is signed more than once. However, Camenisch and Lysyanskaya provide a solution to this as well, by modifying the scheme to be able to sign m by signing an information theoretic hiding commitment to m [58].

To prove possession of a signature in zero-knowledge, first blind the signature σ by choosing random $r, r' \in \mathbb{Z}_p$ and setting $\tilde{\sigma} = (a^{r'}, b^{r'}, c^{r'r}) = (\tilde{a}, \tilde{b}, \tilde{c}^r) = (\tilde{a}, \tilde{b}, \hat{c})$. Then send this blinded signature to the verifier. Both parties now compute $v_x = \mathbf{e}(X, \tilde{a})$, $v_{xy} = \mathbf{e}(X, \tilde{b})$ and $v_s = \mathbf{e}(g, \hat{c})$, and the prover must now show that he knows exponents μ and ρ, such that $v_s^{\rho} = v_x {v_{xy}}^{\mu}$. If this holds, and $\mathbf{e}(\tilde{a}, Y) = \mathbf{e}(g, \tilde{b})$, then the verifier accepts.

2.6 Random Functions

A pseudorandom function (PRF) family is a family of functions indexed by a key k (a random string of length τ bits), and can be designed to have any desired (polynomial in τ) input and output length. The basic property is that even given oracle access to the function (and not the key) it cannot be efficiently distinguished from a truly random function. A pseudorandom function can be constructed from any pseudorandom number generator using the construction given by Goldreich, Goldwasser, and Micali [103].

Related to pseudorandom functions are verifiable random functions (VRF). A verifiable random function is a pseudorandom function, that provides some verifiable property of its output. For example if a random value $r = F_{sk}(x)$ it is possible to prove that r was generated correctly without revealing sk. Verifiable random functions were introduced by Micali, Rabin, and Vadhan [144]. Later Dodis and Yampolskiy constructed a more efficient VRF [89].

Chapter 3

RFID Privacy and Authentication

In this chapter we examine privacy in RFID systems, but these techniques can also be applied in other contexts where a unique identifier is needed. We start by surveying related work in Section 3.1 and in Section 3.2 we define a new model for privacy in RFID systems and discuss some tradeoffs between security and efficiency. This is based on a paper to be presented at the CT-RSA 2008 conference [82].

3.1 Introduction

RFID, as deployed in most areas, uses unique identification and no user control over who gets access to the identity of the tag. The most likely privacy violations will therefore occur through surveillance. The information disclosed will be information about the identity of the tag, but that might, with the help of databases, provide information about the user's activity or his persona. Basically anyone with an RFID reader could learn the identity of the tag, but most likely privacy violations will occur because the identity of the tag is disclosed to organizations, with whom the user might, or might not be familiar.

In this situation there are technical and non-technical mechanisms that can protect privacy. Non-technical measures that can be used include openness, consent and clear definitions of who the data controller is for a specific database with identity codes, for example. Such non-technical solutions are based on legal regulation in the form of legislation and/or voluntary agreements.

However, there are limits to how much protection legal mechanisms can provide. First of all because it becomes a tradeoff between how much an organization can gain by violating the privacy of its users or customers vs. the risk and the cost of being caught. Second, when data are gathered, they are not going to be erased, but the law describing how one is allowed to use these data might change in the future. Finally, just the fact that it is possible for an organization to keep track of a user, has an effect on the privacy experience an individual feels, whether or not any misuse of that information actually occurs.

As previously mentioned, control over our own data in one form or another, is desirable and gives a stronger feeling of privacy, but there are several ways in which the user can be given control over the reading of the identity code.

A mobile phone can, for example, be prevented from transmitting its identity if we turn it off. There are ways to prevent a reader from reading data from an RFID tag, for example by destroying the tag. A tag that does not work, causes no privacy violations, but does not provide any benefits either. A better solution would be to have the tag perform some form of authentication towards the reader, so only authorized readers were allowed to learn the identity of the tag. After the user acquires ownership of the tagged item, he could replace the secret keys used in this protocol, thereby taking control over who can identify the tag.

All in all, the primary method of safeguarding privacy when using global identifiers, is to control which units are authorized to read an identity code. There are many technical measures that ensures that only an authorized reader can learn the identity of a tag, but these solutions require certain features from the units. If we have small devices, such as RFID tags, they will usually not have the computational resources to perform some of the heavy calculations needed for e.g. public key cryptography, and even if it is possible to equip these devices with such capabilities, many will refrain from doing so due to cost issues. Also they will have limited, if any, input devices, making seemingly trivial tasks such as typing in a password impossible. We return to the issue of usability in Chapter 5.

Recently a lot of research has focused on the privacy issue of RFID tags. More specifically, research has been done in the following areas:

Private authentication for RFID tags. The obvious way to prevent many of these privacy problems, is to regulate who can determine the identity of an RFID tag. This is done through a private authentication protocol between the tag and the reader, that discloses information about the identity of the tag, only to the reader which shares key material with the tag. However, designing a private authentication protocol for RFID tags is different from designing a private authentication protocol in general. The primary reasons for this, are the requirements for a small implementation footprint and low computational requirements. This means that only solutions based on symmetric key cryptography have been seriously considered, but symmetric key systems often have the efficiency problem, that a reader must search its entire database of keys, to find a key that matches the tag it is talking to, which clearly does not scale well. Examples of schemes with this property are OSK/AO [14, 15, 154], YA-TRAP [185], and schemes based on so-called hash locks [125, 190].

Molnar, Soppera, and Wagner suggest a private RFID protocol that achieves logarithmic time key-lookup, by using a binary tree of symmetric-keys, where each tag is assigned a key of a path from the root to the leaf [146]. Nohara, Inoue, Baba, and Yasuura propose a scheme with a similar design [153]. While these schemes are more efficient, they do not provide the same level of privacy. The reason is that when two tags share some key material, corrupting one of them gives away some secret information, which can be used to recognize the other tag. We take a closer look at this in Section 3.2. Burmester, Le, and Medeiros use a different approach, where key-lookup is constant for tags that

have not been queried by the adversary. Otherwise the key-lookup is linear [51].

Another technique was proposed by Ateniese, Camenisch, and Medeiros [8]. Their scheme permits re-randomization of an encryption, without knowing the secret key. This could be used to store an encrypted identity code on the tag, which could be transmitted to a reader. The reader would decrypt the identity code, using its private key, re-randomize the encryption, and send it back to the tag. Of course, since the tags are passive, privacy can only be guaranteed when tags talk to honest readers, since a malicious reader could store anything on a passive tag. Furthermore, allowing a tag to be writable many times by anyone could cause other security problems, e.g. when using RFID to protect against counterfeiting.

Given a lightweight one-way trapdoor function, one can convert any symmetric private authentication protocol with linear time key-lookup into a version with a constant time key lookup, simply by encrypting the tag's identity code under the readers public key and thereby telling the reader what symmetric key to use for the actual authentication protocol. An example of such a scheme was proposed by Burmester, Medeiros, and Motta [50].

Since RFID tags have some things in common with another being with limited computational power, namely humans, it has been suggested to use human authentication protocols for RFID tags. The most well known example of this are protocols based on the *Learning Parity in the Presence of Noise* (LPN) problem:

Definition 3.1 (LPN Problem) *Let* $\mathbf{A}$ *be a random* $q \times k$ *binary matrix, let* x *be a random* k*-bit vector, let* $\mu \in (0, \frac{1}{2})$ *be a constant noise parameter, and let* v *be a random* q*-bit vector such that* $|v| \leq \mu q$*. Given* $\mathbf{A}$*,* μ *and* $z = (\mathbf{A} \cdot x) \oplus v$*, find a* k*-bit vector* x' *such that* $|(\mathbf{A} \cdot x') \oplus z| \leq \mu q$

The LPN problem is NP-Hard, but is also believed to be NP-Hard for random instances. A protocol based on this problem, called HB, was proposed by Hopper and Blum [115], and works in the following way:

Assume that a computer C and a human H share a bit vector x of length k. H can now prove this fact towards C as follows: C generates a random k-bit vector a and sends it to H. H generates a random bit v, where $Pr(v = 1) = \mu$, computes the bit $z = (a \cdot x) \oplus v$ and sends it to C. C accepts if $a \cdot x = z$.

Assume for a moment that $\mu = 0$. Then clearly if H knows x this bit will be correct with probability 1, but if he has no information about x, H can only answer correct with probability $\frac{1}{2}$. Hence by repeating this protocol n times, the probability that C will accept in the case where H does not know x is 2^{-n}. The problem is that if a passive adversary captures $O(k)$ valid challenge-response pairs between H and C, he can determine x by gaussian elimination. This is the reason for setting $\mu > 0$. H will intentionally lie with probability μ, and C will accept if fewer than μn responses are incorrect.

HB is not secure against an active adversary. If an adversary pretends to be C and sends the same challenge to the tag multiple times, he can compute the secret x with high probability based on the responses by H. This attack was fixed in a version of the protocol, called HB+, by Juels and Weis [124]. The fix

is for the tag to send an additional blinding vector, which will be added to the response.

The security proof for HB+ does not consider the information leaked by a legitimate prover and legitimate verifier, which allows a man-in-the-middle attack to extract the secret. This attack is described by Gilbert, Robshaw, and Sibert [101]. In 2006 Bringer, Chabanne, and Dottax proposed a protocol called HB++, which they claim to be secure against man-in-the-middle attacks, but it requires additional key material and universal hash functions [48]. In 2008 Gilbert, Robshaw, and Seurin proposed yet another version of HB+ called random-HB#, which is provably resistant to a broader class of attacks [102].

Also in 2006 Fossorier, Mihaljević, Imai, Cui, and Matsuura improved the best known attack against the LPN problem, which significantly increases the number of bits needed for the LPN problem to remain secure [95]. With more complex protocols and increased key lengths, protocols based on the LPN problem have moved away from being human authentication protocols, but might still be suitable for RFID tags.

This is in no way a comprehensive list of RFID authentication protocols. Instead, we refer the reader to the website of Avoine, which he keeps updated with current work on RFID privacy [12]. Also, Juels published a survey of RFID security and privacy [122].

Security models. Several papers have proposed protocols for addressing the privacy problem in RFID systems. However, not much work has addressed formal definitions of security for RFID systems. Juels and Weis propose a definition of what they call *strong privacy* [125], which in turn is based on earlier work by Avoine [13]. Where Avoine aims to classify a range of adversarial capabilities, Juels and Weis tries to describe just one single, but very powerful, adversary, with a simple definition. The most important difference, however, is that the model by Juels and Weis characterizes privacy in systems where tags might share some secrets, whereas the Avoine model does not. We will return to the definition of strong privacy in Section 3.2 and also see why it is important whether tags share secrets or not. In independent work, Burmester, Le, and Medeiros [51] propose a security definition based on Canetti's Universal Composability framework.

Other solutions. Here we briefly take a look at solutions to the RFID privacy problem, that does not fall into the other categories.

The Blocker Tag by Juels, Rivest, and Szydlo [123] works by pretending to be every possible tag at the same time, thus confusing the reader and preventing it from reading the legitimate tag. This ensures privacy, but introduces other problems, e.g. denial of service attacks. Standardized EPC global RFID tags implement a KILL command, which can be used to physically destroy the tag. While this prevents privacy violations, it also removes all the advantages there could be by having an active RFID tag. IBM developed the so-called Clipped Tag, which is an RFID tag where the user can tear off the antenna, thereby reducing the range the tag can be read from, to a few centimeters. This allows

normal use of RFID in the supply chain, but limits the risk of privacy violations after the user acquires the tagged item. Unfortunately it shares the problem with the KILL command, that in many after-sale applications, the tag is useless.

Finally we have the RFID Guardian which is a mobile battery-powered device that offers personal RFID security and privacy management [166]. It requires readers to authenticate themselves before being allowed to read a particular RFID tag. If the reader fails authentication, the RFID Guardian broadcasts jamming signals to confuse the reader, just as the Blocker Tag does. In this way the RFID Guardian enforces the privacy policy chosen by the user.

3.2 Tradeoffs between Privacy and Efficiency

As mentioned earlier, much work has focused on solving the security issues of RFID systems, yet not much work has been done trying to define security for such systems. In 2006 Juels and Weis proposed a definition of what they call *strong privacy* [125]. Strong privacy is indeed a strong notion, primarily because the adversary is given a lot of power: He can corrupt any number of tags (but not the reader) and read their contents, he can eavesdrop and schedule the tag/reader communication any way he wants, and he can himself select the tags whose privacy he wants to break. The work of Juels and Weis only addresses privacy, that is, making sure that the communication of a tag does not allow an external adversary to determine the identity of the tag. Of course, another natural requirement is that a reader should be able to determine whether the tag it reads is valid and not fabricated by an adversary, for instance. Indeed, if this was not required, tags could just return random information all the time or just not reply at all. This would trivially be private, but would of course lead to a useless system.

In this section, we propose an extension to the strong privacy definition so one can also require completeness and soundness, with the intuitive meaning that the reader accepts valid tags and valid tags only. More specifically, soundness in the weakest sense means that we assume the adversary cannot corrupt tags, and when the reader accepts an instance of the read protocol, a (uncorrupted) tag has been involved in that instance at some point. So in this weak flavor, it is not required that the reader knows which tag it has been talking to. We also suggest a stronger version where corruptions are allowed and the reader must output the identity of the (honest) tag that was involved.

The concept of strongly private and sound systems is closely related to existing concepts for anonymous identification schemes, such as identity escrow schemes [129] or group signature schemes [9, 10, 33, 59, 71, 128], which we will briefly review when we talk about anonymous authentication in Chapter 4 and 6. They are not the same, however: Our model is designed to model RFID systems, and where identity escrow and group signature schemes are by definition public-key techniques, we want to cover techniques based on secret-key algorithms only.

The most important privacy issue regarding RFID tags is the issue of being able to systematically track individuals as they carry RFID enabled goods from the supermarket, embedded in the their clothes, etc. In this scenario, it is

reasonable to assume that the adversary cannot himself choose the tags he wants to track. Strong privacy is therefore more than we need in this scenario, so we introduce a weaker, but more suitable, definition called *benign-selection privacy*.

Juels and Weis suggest a system that satisfies their definition, building on earlier work by Weis, Sarma, Rivest, and Engels [190]. In this scheme, each tag is given an independently chosen key, and the reader must search exhaustively through all keys every time a tag is read. This of course does not scale well, but Juels and Weis conjecture that this is, in a certain sense, unavoidable: In strongly private systems that use only symmetric cryptography, and where tags are independently keyed, the reader must access all, or at least a large fraction of the keys in the system. Here, we prove this conjecture. We need to assume that the system is complete and sound, but this is of course a natural requirement and is necessary anyway to exclude degenerate cases, such as when tags only send random information.

The limitation to symmetric cryptography is clearly necessary: With public-key technology, a tag could send its identity encrypted under the reader's public-key, and then prove its identity using some shared-key technique, for instance. This does not require the reader to look at any information that is not related to the relevant tag. There has in fact been recent work in the direction of implementing public-key on very small devices [22], but even if public-key enabled RFID tags are only slightly more expensive than symmetric-key only tags, this will still inhibit the use of public-key technology in large scenarios that require millions of tags, in order to maximize profit. Therefore we believe the question of what can be done with symmetric techniques is of interest, both theoretically and in practice.

The limitation to schemes with independent keys is not surprising. It follows from work by Molnar, Soppera, and Wagner [146] that when dependent keys are allowed, we can have a system where the reader only needs to look at a logarithmic (in the number of tags) number of keys. This comes at the price that strong privacy only holds if the adversary is "radio-only", i.e., he does not corrupt any tags. If the adversary corrupts even a single tag, strong privacy is lost, and benign-selection privacy is lost with large probability. This makes it natural to ask if there are alternative solutions where we can get some amount of privacy with a larger number of corruptions without going back to systems where the reader does exhaustive search over all keys.

In this section, we first argue that for a wide range of RFID systems, there has to be a tradeoff between the efficiency of the reader and the resources we can allow the adversary to have. We then propose a class of protocols offering a new range of tradeoffs between security and efficiency. For instance, the number of keys accessed by a reader to read a tag can be significantly smaller than the number of tags while retaining soundness and privacy, as long as we assume suitable limitations on the adversary.

3.2.1 Model and Definition

Juels and Weis define *strong privacy* for RFID systems using a model of which we give a summary here, for details refer to [125].

The system consists of tags $\mathcal{T}_i, i = 1..n$ and a reader $\mathcal{R}$. For simplicity, we assume that there is only one reader. Tags can receive SetKey messages which will cause the tag to reveal its secret key, and the caller may then send a new key to the tag. This can be used to initialize the system and also to model an attacker corrupting a tag to learn its key. A tag may receive a (TagInit, *sid*) message (where *sid* is a session id), which is used in the start of a session. The tag will forget any previous value of *sid*, so a tag may only run a single session at a time. Finally, the tag may respond to a protocol message c_i, called a challenge in [125], by a response r_i. A protocol may consist of several rounds of challenges and responses.

A Reader may receive ReaderInit messages, causing it to generate a fresh session identifier *sid* and a first protocol message c_0 to be sent to a tag. It may also receive pairs of the form (sid, r_i). It will then return either a new message c_{i+1} to be sent to the tag or *Accept* or *Reject*. In [125], a reader, if it returns *Accept*, is not required to say which tag it thinks it has been talking to. We assume here that it may also return the identity of a tag. The reader keeps an internal log of all challenge and response pairs for each session id that is active, and decides based on this whether to accept or reject. A reader may be involved in several sessions simultaneously, but its behavior in a session only depends on messages it receives in that session and the fixed key material it holds.

We allow the adversary $\mathcal{A}$ to schedule all messages as it wants, and generate its own messages. The adversary is parameterized as follows: r is the number of ReaderInit messages it generates, s is the number of computational steps and t is the number of TagInit messages it generates. Finally, k is a cryptographic security parameter. Juels and Weis do not treat the number of SetKey messages, i.e., the number of corrupted tags, as a separate parameter, but simply say it has to be at most $n-2$. As we shall see, however, the number of corrupted tags is a very important parameter, so we will define u to be the number of tags corrupted by the adversary. A summary of these parameters can be found in Figure 3.1. Note that this model also captures an adversary that passively listens to a session between reader and tag, namely he starts a session with the reader and one with the tag and simply relays messages between the parties.

k: security parameter	n: number of tags in the RFID system $\mathcal{S}$
r: number of ReaderInit messages allowed	s: number of computational steps allowed
t: number of TagInit messages allowed	u: number of SetKey messages allowed

Figure 3.1: Description of parameters

The system is initially setup by running a probabilistic key generation algorithm $\text{Gen}(1^k)$ which produces a set of keys $key_1, ..., key_n$ to be assigned to the tags. Of course, $\mathcal{A}$ does not know these keys initially.

Let $\mathcal{S} = (\text{Gen}, \mathcal{R}, \{\mathcal{T}_i\})$ denote an RFID system. Strong privacy is defined

via an experiment called $\mathbf{Exp}^{priv}_{\mathcal{A},\mathcal{S}}[k,n,r,s,t]$. Here, we run the system where the adversary may corrupt tags, initiate sessions, etc., observing the limitations put on him. This ends by the adversary selecting two uncorrupted tags, called $\mathcal{T}_0^*, \mathcal{T}_1^*$. He is then given oracle access to $\mathcal{T}_b^*$ where b is a random bit. He may now again corrupt other tags and initiate sessions, and must finally guess the value of b. However, we have to assume that in this last phase, when using the reader to interact with $\mathcal{T}_b^*$, he only learns whether the reader outputs accept or reject and not the identity found by the reader. Otherwise, he could just let the reader identify $\mathcal{T}_b^*$. The system is said to be (r,s,t)-private if any (r,s,t)-adversary's advantage over $1/2$ in guessing b is negligible as a function of k. We propose here to define also (r,s,t,u)-privacy, which is the same, except that the adversary may only corrupt at most u tags. However, for some systems, the advantage that can be achieved depends not only k, but on all the parameters, and does not tend to 0 as we increase k, if other parameters are constant. We will therefore use a variant of strong privacy here:

Definition 3.2 *Strong (k,r,s,t,u,n,ϵ)-privacy is defined via the experiment $\mathbf{Exp}^{priv}_{\mathcal{A},\mathcal{S}}[k,n,r,s,t,u]$ which is the same as Juels and Weis' except that the adversary can only corrupt up to u tags. We say that the system is strongly (k,r,s,t,u,n,ϵ)-private if any adversary observing the limitations in the experiment has advantage at most ϵ.*

Experiment $\mathbf{Exp}^{priv}_{\mathcal{A},\mathcal{S}}[k,n,r,s,t,u]$ **Setup:**

1. $\textsc{Gen}(1^k) \rightarrow (key_0, ..., key_n)$
2. Initialize $\mathcal{R}$ with $(key_0, ..., key_n)$
3. Set each $\mathcal{T}_i$'s key to key_i with a SetKey call

Phase 1 (Learning):

4. $\mathcal{A}$ may do the following in any interleaved order:
   (a) Make ReaderInit calls, without exceeding r overall calls
   (b) Make TagInit calls, without exceeding t overall calls
   (c) Make SetKey calls, without exceeding u overall calls
   (d) Send challenges and responces to tags and reader respectively, without exceeding s overall steps

Phase 2 (Challenge):

5. $\mathcal{A}$ selects two tags $\mathcal{T}_i$ and $\mathcal{T}_j$ to which it did *not* send SetKey messages
6. Let $\mathcal{T}_0^* = \mathcal{T}_i$ and $\mathcal{T}_1^* = \mathcal{T}_j$ and remove both of these from the current tag set
7. Choose a random bit $b \in \{0,1\}$ and provide $\mathcal{A}$ access to $\mathcal{T}_b^*$
8. $\mathcal{A}$ may do the following in any interleaved order:

(a) Make ReaderInit calls, without exceeding r overall calls

(b) Make TagInit calls, without exceeding t overall calls

(c) Make SetKey calls, without exceeding u overall calls to any tag in the current tag set

(d) Send challenges and responces to tags and reader respectively, without exceeding s overall steps

9. $\mathcal{A}$ outputs a guess bit b'

$\mathcal{A}$ succeeds if $b = b'$

As Juels and Weis note in [125], strong privacy may be too strong a notion for many real world applications. In particular, the adversary can freely choose the target tags he wants to be challenged on. He may not have that much power in real life, where the choice may be forced on him by the environment he operates in. One may try to model this by having the target tags be chosen from some distribution independently of the adversary – this idea is already present in the work of Avoine [13]. But it is very difficult to single out a distribution that realistically models the environment. We therefore propose a new model called *benign-selection privacy* where we allow any distribution as long as it only selects uncorrupted tags.

Definition 3.3 *Benign-selection privacy is defined via an experiment called* $\mathbf{Exp}^{bspriv}_{\mathcal{A},\mathcal{S},\mathcal{D}}[k,n,r,s,t,u]$ *which is the same as* $\mathbf{Exp}^{priv}_{\mathcal{A},\mathcal{S}}[k,n,r,s,t,u]$*, except that the adversary does not select the two tags* $\mathcal{T}_0^*, \mathcal{T}_1^*$*. Instead they are chosen at random from distribution* D *among all uncorrupted tags. We think of* D *as a probabilistic algorithm that only gets the set of corrupted tags as input, and outputs the index of the target tags, i.e., the choice is uncorrelated to adversarial activity other than corruptions. We say that the system is* (k,r,s,t,u,n,ϵ)*-private with benign* D*-selection if any adversary observing the limitations in the experiment has advantage at most* ϵ*.*

In the following, it will often be cumbersome and unnecessarily complicated to specify s, the number of computational steps, exactly. We will often replace s by a $poly(k)$, meaning that the statement involved holds for any adversary that uses time polynomial in k.

It is natural to expect a system as described here to also have the properties that valid tags are accepted, and that the adversary cannot impersonate a tag unless he corrupts it. This aspect was not treated in [125] (but was also not the main goal there). We propose to define this as follows:

Completeness Assume that at the end of session *sid* the internal log of the reader $\mathcal{R}$ for that session contains pairs (c_j, r_j) where all r_j's were generated by an honest tag in correct order. Completeness means that $\mathcal{R}$ outputs Accept with probability 1 for any such session.

Strong Soundness Consider the following experiment similar to the privacy experiment of Juels and Weis:

Experiment $\mathbf{Exp}^{sound}_{\mathcal{A},\mathcal{S}}[k, n, r, s, t, u]$:

Setup:

1. $\textsc{Gen}(1^k) \rightarrow (key_0, ..., key_n)$
2. Initialize $\mathcal{R}$ with $(key_0, ..., key_n)$
3. Set each $\mathcal{T}_i$'s key to key_i with a SetKey call

Attack:

4. $\mathcal{A}$ may do the following in any interleaved order:
   (a) Make ReaderInit calls, without exceeding r overall calls
   (b) Make TagInit calls, without exceeding t overall calls
   (c) Make SetKey calls, without exceeding u overall calls
   (d) Send challenges and responces to tags and reader respectively, without exceeding s overall steps

Let E be the event that occurs if $\mathcal{R}$ at some point outputs $(Accept, i)$ at the end of session sid where $\mathcal{T}_i$ is not corrupted, yet $\mathcal{R}$'s internal entry for sid only contains pairs (c_j, r_j) where $\mathcal{T}_i$ was not involved in generating r_j. We say that the system provides strong (r, s, t, u)-soundness if the probability that E occurs is negligible in k.

Weak Soundness Weak (r, s, t)-soundness is defined by the same experiment as above, except that $\mathcal{R}$ now only has to output Accept or Reject at the end of a session, $\mathcal{A}$ is not allowed to corrupt tags, and the error event E is now defined to be that $\mathcal{R}$ outputs Accept, and yet no tag has been involved in the session.

3.2.2 Independent Keys

As mentioned earlier, our goal in this section is to prove the speculation by Juels and Weis: In any strongly private, complete and sound RFID system, the reader must access a key for every tag, or at least a large fraction of them, when reading a tag. This can only be expected to hold, however, when keys for different tags are independently chosen, and the system "only" uses symmetric cryptography. If public-key cryptography was allowed, a tag could first encrypt its identity under the reader's public-key, and then show possession of some secret that is shared between reader and this tag only.

To prove something, we need to formalize the constraints on the system. For the independence of keys, this is easy, we simply assume that each tag $\mathcal{T}_i$ gets a key K_i chosen independently from all other keys by a key generation algorithm G_i, i.e., $K_i \leftarrow G_i(1^k)$ where k is the security parameter. As for the constraint that "symmetric cryptography and nothing else is used", we will give the system access to a pseudorandom function, $\phi_\cdot(\cdot)$, and we will assume that every key K_i in the system is used only as a key to this function, i.e., tag $\mathcal{T}_i$ or reader use $\phi_{K_i}(\cdot)$ as a black box. This means that we can equivalently give

tags and reader oracle access to $\phi_{K_i}(\cdot)$ for any key they need to use. Therefore, when in the following we say that "the reader accesses a key", this means it calls the oracle that holds that key.

Now, to model that the pseudorandom functions are the essential cryptographic resource used, we will simply assume that the keys $\{K_i\}$ held by the reader and tags are the only secret data in the system. More precisely, we think of the reader's algorithm as an interactive Turing machine that takes no private input, but may make oracle calls to $\phi_{K_i}(\cdot)$ for any K_i. Similarly, a tag may only call its own pseudorandom function, whereas the adversary may only call $\phi_{K_i}(\cdot)$ if he has corrupted $\mathcal{T}_i$. We will say that such a system is *essentially symmetric*.

Note that an essentially symmetric system is not prevented from using public-key, or using secret-key techniques in a non-blackbox way – the reader could try to do a Diffie-Hellman key exchange with a tag, for instance, or generate a key for a pseudorandom function and use this key in any way it wants. Nevertheless, the constraints we have defined are sufficient to show what we are after. To get better intuition for why this is the case, one may note that, while the reader is free to generate a public encryption key and send it to a tag, the tag cannot immediately verify that the key comes from the reader and not the adversary. Thus it would not be secure to send the tag's id encrypted under the public key.

The first lemma formalizes the straightforward intuition that if keys are independent, a reader cannot determine if it is talking to a valid tag unless it accesses the key for that tag. More formally:

Lemma 3.1 *Consider an RFID system that is complete, weakly* $(1, poly(k), 0)$*-sound, and uses independent keys. Consider a session between reader and a tag where the adversary does not modify the traffic. In any such session, the algorithm executed by the reader when reading a tag* $\mathcal{T}_i$ *will access* K_i*, except with negligible probability.*

Proof. We consider all probabilities as taken over the choice of keys and the random coins used by tag and reader in the session. Let E be the event that the reader does not access ϕ_{K_i}. By completeness, the reader should accept with probability 1, so the probability that the reader accepts and E occurs equals $Pr(E)$. Assume for contradiction that $Pr(E)$ is non-negligible. Then an adversary could fabricate his own tag $\mathcal{T}_i'$ with a key K_i' generated by G_i, and start a session between this tag and the reader, while simply following the protocol. Now by independence of keys, as long as E occurs, conversations with $\mathcal{T}_i'$ and $\mathcal{T}_i$ are perfectly indistinguishable. Hence, the reader accepts with probability at least $Pr(E)$, which contradicts weak soundness. □

The next theorem uses the observation that in an essentially symmetric system, the *only* difference between the honest reader and an adversary is that the reader has access to all keys, while the adversary initially does not. He can, however, corrupt tags and get access to (some of) the keys. He can therefore potentially run the same algorithm that the reader uses when reading a tag.

Theorem 3.1 *Assume an essentially symmetric RFID system is complete and weakly* $(1, poly(k), 0)$*-sound. Assume also that the reader algorithm accesses at*

most αn of the keys, for a constant $\alpha < 1/2$. Such a system cannot have strong $(k, 0, poly(k), 1, \alpha n, n, 1/2 - \alpha)$-privacy

Proof. We describe an adversary that will break strong privacy for any system that is complete and weakly sound and where only αn oracles are accessed. The adversary picks uniformly a pair of tags $\mathcal{T}_i, \mathcal{T}_j$, and uses these two as the challenge pair $(\mathcal{T}_0^*, \mathcal{T}_1^*)$ from the strong privacy definition. It then gets oracle access to $\mathcal{T}_b^*$, where $b = 0$ or 1 and should try to guess which of the two it is talking to. To do this, it executes the read protocol with $\mathcal{T}_b^*$, and while doing so, it emulates the reader's algorithm. Whenever the reader algorithm wants to access K_t, the adversary corrupts $\mathcal{T}_t$, and may now call the pseudorandom function with key K_t. This goes on until the reader algorithm wants to access K_t where $t = i$ or j. In this case the adversary outputs 0 if $t = i$ and 1 otherwise and then stops.

To analyze the probability that this adversary has success, suppose, for instance, that $b = 0$. Since our adversary follows the protocol when talking to $\mathcal{T}_b^*$, we can apply Lemma 3.1 to conclude that the reader will access K_i when talking to $\mathcal{T}_b^*$ with probability essentially 1. On the other hand, the probability that it will not access K_j is greater than $1-\alpha$ because only αn keys are accessed (one of which is K_i), and given i, j is uniform over all values different from i. It follows that the adversary's guess is correct with probability $1 - \alpha$ which is a constant greater than $1/2$ and hence we contradict strong privacy. □

Note that since the adversary we construct in the proof selects target tags uniformly, this same argument also shows that a system as specified in the theorem cannot even have benign D-selection privacy where D is the uniform distribution.

One might use some form of pre-computation to perform key lookups more efficiently. For example Avoine, Dysli, and Oechslin [14, 15] propose to use Hellman tables [113] in the protocol of Ohkubo, Suzuki, and, Kinoshita, to reduce key lookup time to $O(n^{2/3})$ at the cost of using an additional $O(n^{2/3})$ space [154]. Since the construction of the table requires accessing all keys in the system, methods using Hellman tables do not immediately contradict the lower bound. We can, however, argue that such methods cannot provide both soundness and privacy: To initialize such a table one must predict all possible outputs from the tag, which in turn means that the tag can only have a fixed number of outputs, m. Juels and Weis show how to break strong privacy for such a scheme, simply by querying a tag m times, and use the reader to distinguish it from another tag that has been queried less than m times [125]. Note that the reader can only accept having the same conversation once with a tag, otherwise a simply replay attack could break the soundness.

3.2.3 Correlated Keys

We have shown in the previous section that if we want strong privacy and tags have independent keys, the reader has to access least half of the keys in the worst case. This obviously does not scale well, so we now look at how much

privacy and soundness we will loose in return for efficiency if we allow the keys to be correlated.

It was already known from the work of Molnar, Soppera, and Wagner that using correlated keys, one can obtain the property that the reader only needs to access a logarithmic number of keys [146]. Unfortunately, this comes at the price that strong privacy is lost already if the adversary corrupts a single tag. This is due to the fact that the system works with a pair of keys (K_0, K_1), where half the tags hold K_0, the other half hold K_1 – as well as many other keys, arranged in a tree structure, which is not important here, however. Corrupting a single tag tells the adversary one of the keys, say K_0. The protocol is such that one can easily extract from the responses tags give, a part that is computed only from K_0 or K_1. This gives the adversary a way to compute from the responses of an uncorrupted tag which of the two keys it holds. Since half the tags hold K_0, 2 sessions with random chosen tags will locate two tags holding different keys with probability 1/2. Clearly, using two such tags as the target in the privacy experiment, the adversary can identify with certainty which tag he talks to. It is not even private with benign selection, no matter which distribution is used: the distribution is by definition independent of which keys are held by uncorrupted tags, so we again have that the target tags hold different keys with probability 1/2. Of course, an error probability of 1/2 is too large in practice.

This makes it natural to ask if we can get privacy with a larger number of corruptions without going back to systems where the reader does exhaustive search over all keys.

3.2.3.1 A Necessary Tradeoff

First, it is useful to observe that in the kind of systems we look at here, some tradeoff between efficiency of the reader and privacy is unavoidable: suppose the key generation algorithm works by generating independently a number of keys, and then assigning to each tag a subset of these keys. The system we propose below, as well as the systems proposed by Molnar et al. and by Juels and Weis, are all of this type.

Let K be one of the keys used. We will say that K is *efficiently decidable* if there is an efficient algorithm that, when given K and a session between a tag $\mathcal{T}$ and the reader, can decide whether $\mathcal{T}$ holds K or not. For instance, it may be that the tag, if indeed it holds K, computes a particular part of its response only from K. One can then from K compute what the tag should say if it knows K and compare to what it actually said. In the systems from [125, 146], all keys are efficiently decidable.

An efficiently decidable key can be used by the reader towards identifying the tag it is reading, because it can tell whether the tag is in the set of tags that know K or in the complement. However, such a key can also be used by the adversary, who may learn K by corrupting a tag, and can now also distinguish tags that know K from those who do not. Clearly, if the adversary can locate two tags, of which one holds K and the other doesn't, then he can break strong privacy. Let $p(K)$ be the number of tags that hold the key K. The case where $p(K) = n/2$ is the case where the reader gets maximal information

from knowing K, namely one bit of information on the identity of the tag. Unfortunately, this is also the optimal case for the adversary, since interactions with a constant number of tags will be sufficient to locate two target tags that can be used to break the privacy.

One may treat this problem either by letting every part of the tag response depend on several keys, or make sure that $p(K)$ is small for every efficiently decidable key K. Both approaches make life harder for the adversary as well as for the reader. We give below an example of the second approach.

3.2.3.2 A Tradeoff Construction

Our construction depends on two parameter, v, c. Typically, v will be quite large, say $v = n^d$ for some constant $d < 1$, while c may be something small, say constant or logarithmic in n. We will assume that we have a pseudorandom function $\phi.(\cdot)$. It is straightforward to construct such functions from a cryptographic hash function by simply hashing the key together with the input, this is provably secure in the random oracle model. Other constructions based on, e.g., AES are also possible.

The key generation involves generating c lists of keys to the pseudorandom function ϕ, $K^j = (k_1^j, k_2^j, ..., k_v^j)$ for $j = 1..c$.

$$
\begin{aligned}
K^1 &= k_1^1, \boxed{k_2^1}, k_3^1, k_4^1, \ldots, k_v^1 \\
K^2 &= \boxed{k_1^2}, k_2^2, k_3^2, k_4^2, \ldots, k_v^2 \\
K^3 &= k_1^3, k_2^3, \boxed{k_3^3}, k_4^3, \ldots, k_v^3 \\
&\ldots \\
K^c &= k_1^c, \boxed{k_2^c}, k_3^c, k_4^c, \ldots, k_v^c \quad \boxed{k_i}
\end{aligned}
$$

Figure 3.2: Example: Keys assigned to a tag $\mathcal{T}_i$ with string $str_i = (2, 1, 3, \ldots, 2)$

We assign to each tag $\mathcal{T}_i$ a random string $str_i = (s_{i,1}, ..., s_{i,c}) \in Z_v^c$, c keys $(k^1_{s_{i,1}}, ..., k^c_{s_{i,c}})$, and a key k_i that is unique to $\mathcal{T}_i$ (see Figure 3.2). The probability that two tags will be assigned the same string is at most n^2/v^c, we assume v, c are chosen such that this is negligible. Let n_T, n_R be nonces chosen by tag, respectively reader, such that these values do not repeat. Then the protocol between the tag $\mathcal{T}_i$ and reader is:

1. $\mathcal{R}_i \longrightarrow \mathcal{T}_i$: n_R
2. $\mathcal{R} \longleftarrow \mathcal{T}_i$: n_T, $\phi_{k_{s_{i,j}}}(n_T, n_R)$, for $j = 1, .., c$, and $\phi_{k_i}(n_T, n_R)$. The intuition is that the first c values allow the reader to identify the tag, while the final value proves that the tag is who it claims to be.

For the j'th pseudorandom function value received, $j = \{1 \ldots c\}$, the reader searches through the v keys in K^j and checks if one of these will generate the

value received, i.e., for each $k \in K^j$ one checks if $\phi_k(n_T, n_R) = \phi_{k_{s_{i,j}}}(n_T, n_R)$. If this is not the case, reject and stop. Otherwise note the index of the key. The indices noted form a string $(s_1, .., s_c)$. If this string matches the string assigned to some tag $\mathcal{T}_i$, and the final pseudorandom value received is equal to $\phi_{k_i}(n_T, n_R)$, output (accept, i). Else output reject.

To show security of the system, we first go to the *independent oracles model*, i.e., we replace each call to ϕ using key k by a call to a random oracle O_k, using independent oracles for different keys. The adversary can only call an oracle O_k if he corrupts a tag that holds k.

It is straightforward to see that if we model the hash function used in the proposed construction of ϕ by a random oracle, then an adversary playing the privacy or soundness game is exactly working in the oracle model just described. For this reason and for simplicity, we will analyze the system in this model

The first result on our system shows that, without loss of generality, we may consider only adversaries who do no talk to the reader:

Lemma 3.2 *In both the privacy and soundness games, sessions that the adversary initiates with the reader can be simulated without access to the reader, but with access to those oracles that the adversary can access. The simulation is perfect, except with probability negligible in k.*

Proof. We describe an algorithm for simulating the sessions in question: In any session, the reader first sends a nonce n_R, this can be simulated by simply following the reader's algorithm for selecting nonces. The message that the adversary returns must consist of a nonce n_T and $c+1$ values $r_1, ..., r_c, s$. Note that the reader checks these values against oracle outputs generated from the fresh input n_R, n_T, and that we may assume that oracle answers are sufficiently long so they cannot be guessed except with negligible probability. For these reasons, the adversary can only hope to have the reader accept if he generated each of the $c+1$ response values by either using an oracle he has direct access to, or by starting a session with an uncorrupted tag and using (part of) the tag's response. If this is not the case, we can return $reject$ to the adversary: in real life the reader will reject such a response except with negligible probability. But if the adversary has indeed generated the entire response by calling oracles (directly or indirectly), we know the identity k' of the oracle that generated the last value in the response. If the call to oracle $O_{k'}$ was made by an uncorrupted tag $\mathcal{T}_j$, this has to be because that tag received n_R as a challenge and therefore produced a correct response for nonces n_R, n_T. If we see that the adversary forwards this response to the reader, we return $(accept, j)$ as the real reader would have done. If the adversary has replaced any of the first c values with other oracle responses, we return $reject$, which is correct except with negligible probability.

The only remaining possibility is that it was the adversary who called $O_{k'}$. This means he must have corrupted the tag $\mathcal{T}_i$ giving access to this oracle, and so he also has access to to the other c oracles that this tag possesses. Therefore, having generated the message sent to the reader, we can check whether this is a correct response from $\mathcal{T}_i$. If this is not the case, we return reject to the adversary. Otherwise, we return $(accept, i)$. □

The following lemma turns out to be essential for privacy:

Lemma 3.3 *Consider an adversary that does not start any session with the reader. Let M be the set of oracles that the adversary gains access to during the privacy game. Let E be the event that the following condition is satisfied after the game: the adversary has started at least one session with some uncorrupted tag $\mathcal{T}$, and one of the oracles assigned to $\mathcal{T}$ is in M. In the privacy game, by convention, the adversary selecting the two target tags counts as starting a session with both tags. Let t' be the number of different tags the adversary talks to during the game. The probability that E occurs is at most*

$$\frac{ct'u}{v} + \frac{ct'u}{v-u}$$

Proof. Suppose we are at some point in the game where E has not occurred yet. This means that for all uncorrupted tags the adversary has talked to, he knows that they only have oracles he has no access to, but due to the randomness of the oracles, he has no information on their identity.

The adversary may now start a session with a new tag he did not talk to before, or corrupt a tag. For each of these moves, we bound the probability that E will occur after the move:

Start New Session Since the adversary has not previously talked to the tag $\mathcal{T}_i$, given what he knows, str_i is uniform. We can therefore model what goes on as follows: look at one of the c positions in str_i, and let $x \in Z_v$ be the number in this position. Now, x is uniform over v possibilities, and the adversary has success, if x happens to be one of the $\leq u$ values corresponding to oracles he can access. So the adversary has success in one position with probability at most u/v, and therefore has success in any position with probability at most $\frac{cu}{v}$

Corrupt New Tag For the previously uncorrupted tag $\mathcal{T}_i$, consider again x, the number at some position in str_i. Then given what the adversary knows, before he corrupts $\mathcal{T}_i$, x is uniform over at least $v-u$ possibilities, if the adversary talked to $\mathcal{T}_i$ before, he knows x does not match any of the $\leq u$ possibilities he knows from already corrupted tags. The adversary hopes x will hit one of the $\leq t'$ possibilities for tags he talked to, so the probability of success is at most $t'/(v-u)$ for one position and $\frac{ct'}{v-u}$ for all positions.

Finally, since there are at most t' respectively u steps that could cause the first respectively second kind of event, the lemma follows. □

We are now ready to prove security of our construction.

Theorem 3.2 *If the hash function used in the construction above is modeled by a random oracle, then the RFID system is $(poly(k), poly(k), poly(k), n)$-strongly sound, and is strongly $(k, r, poly(k), t, u, n, \epsilon)$-private, where*

$$\epsilon = \frac{ctu}{v} + \frac{ctu}{v-u} + negl(k)$$

and where $negl(k)$ is a negligible function of k.

Proof. Completeness is obvious from the fact that the strings assigned to tags are unique except with negligible probability.

For soundness, recall that the adversary wins the soundness game if a session is generated where the reader outputs $(accept, i)$, but the (uncorrupted) tag $\mathcal{T}_i$ did not participate. Since the input nonces are fresh and oracles answers cannot be guessed in advance except with negligible probability, the oracle O_{k_i} must have been called to generated the last part of the response. But this is impossible since $\mathcal{T}_i$ did not participate and the adversary does not have access to O_{k_i} as long as $\mathcal{T}_i$ is uncorrupted.

Finally, for privacy, note that by Lemma 3.2, any adversary $\mathcal{A}$ playing the privacy game can be replaced by a new adversary $\mathcal{A}'$, who does not start sessions with the reader, and such that the advantage of $\mathcal{A}'$ is smaller than that of $\mathcal{A}$ by at most a negligible amount. This, together with Lemma 3.3 immediately implies the privacy result. □

Finally, we show that the adversary's advantage in the benign selection privacy game is much smaller:

Theorem 3.3 *Our system is $(k, r, poly(k), t, u, n, \epsilon)$-private with benign selection for any distribution D, and where $\epsilon = 2cu/v + negl(k)$*

Proof. As above, we can assume that the adversary does not talk to the reader, at the cost of adding a negligible amount to the advantage. Now consider the situation when the target tags are chosen. For each of the c positions in the strings assigned to tags, the adversary can access at most u of the v oracles assigned to this position. Hence, when an uncorrupted tag is chosen, no matter how this is done, the probability that its oracle for this position is known to the adversary is u/v since "names" of tags are assigned uniformly and independently. Since the two target tags hold a total of $2c$ oracles that could be used to distinguish them, the probability that at least one of them is known to the adversary is at most $2cu/v$. On the other hand, if the adversary has no oracles in common with the target tags, he cannot distinguish them at all. □

3.2.4 Efficiency

The interest in this result is that it shows a possibility for a new tradeoff between security and efficiency for large systems, where the adversary can be expected to only corrupt or talk to a number of the tags that is very small compared to the total number of tags in the system. More precisely, for parameter values such that $r, t', u << v << n$, but still $n^2 < v^c$. However, for particular values of r, t', u and c, v and hence n must very large to make the privacy advantage be small. This has to do with the fact that we are asking for strong privacy and this is a very strong demand. Below, we show that the systems performs much better under the privacy definition with benign selection. On the practical side, note that the reader needs to look at only cv keys which can be much smaller than n. Also, each tag only has to hold $c + 1$ keys. Although the total number of keys in the system is greater than n, this does not mean that the reader has

to store this many keys – they can be generated pseudorandomly from a single key when they are needed.

Let us look at a concrete example of parameters in the benign selection model for any distribution. Suppose we choose $v = 2^{16}$ and $c = 4$. Then we can accommodate over 33 million tags, say $n = 2^{25}$, and each tag only needs to store 5 keys. If the adversary can corrupt 100 tags, the above says that his chance of distinguishing two tags that are chosen for him is at most 1/100. Note that even if the adversary is lucky with one pair of tags, his chance against another pair is still only 1/100, so we think this can be quite reasonable in practice. In other words, even though a probability of 1/100 is not negligible in the usual sense, this is not necessary, if the "bad event" does not imply a complete break of the system. With these parameters, the reader must search through at most 2^{18} keys to identify a tag, which is clearly better than 2^{25}, which was needed to get strong privacy. We can even increase n without increasing the number of keys to search through, as long as we keep the probability that two tags will be assigned the same key n^2/v^c reasonably small.

Chapter 4

Anonymous Authentication

In this chapter we look at some cryptographic techniques for anonymous authentication. First we survey related known techniques in Section 4.1 and then we introduce a new concept we call *Unclonable Group Identification* in Section 4.2. This is based on a paper presented at the Eurocrypt 2006 conference [80].

4.1 Introduction

In Chapter 3 we looked at privacy in RFID systems, where the primary concern was revealing the tag's identity to unauthorized readers. The solutions we saw were based on only revealing the tag's identity to readers who shared a secret key with the tag, but this is not very flexible. If multiple authorities need to be able to read the same RFID tag, they are able to link sightings of that particular tag. Furthermore, users have limited choices in protecting their privacy. If one wants to use RFID for user authentication (ignoring for a moment that the identity of a tag might be easy to spoof) he can either choose to provide the identity, which will be linked to a database containing all information registered about him, or he can choose not to disclose it. There is no option to prove just the single fact he wants to prove about his identity, for example that he is allowed to access a specific resource, without revealing everything.

In general privacy problems are likely to occur when an identification is not context dependant, that is when a unique identity is shared across different domains. This problem does not only apply to RFID tags, but to many other technologies as well. In Denmark, for example, every citizen has a so-called CPR number. This number is used to identify individuals everywhere from government institutions, to the bank, and to the local video rental store. This is convenient since it is a unique way to identify a citizen, and we all have some form of ID card with this number printed on, but from a privacy point of view it is an extremely bad idea. There are many other examples of unique identifiers being used: Usernames, e-mail addresses, digital signatures, etc. We don't solve all privacy problems simply by showing these credentials to several organizations in a private way. While it does protect against outsiders eavesdropping on the protocol, it does not protect against insiders from these organizations.

However, there are no reasons to use unique identification in all contexts.

There are technical solutions that ensure a certain degree of anonymity while still having the same advantages as with a unique identifier. These solutions can protect a user's privacy by, for example, putting him in control of who can access his personal information, providing anonymity, making his different actions unlinkable, not leaving electronic footprints behind, etc. The thing that all these solutions have in common is that they usually demand more: Both of the unit doing the computation, but often also of the user, due to the increased technical complexity.

This is often a problem in connection with pervasive computing, where many units have limited computing capacity, meaning that traditional cryptographic algorithms can not always be used, simply because they take too long, or use too much memory. In Chapter 6 we take a look at how this affects certain scenarios and how that might be solved in the context of digital signatures, but in this chapter we assume that we are not severely limited by the computational resources available. This makes sense when we are developing solutions that will run on more expensive hardware.

Digital Signatures. As mentioned in Chapter 2, digital signature schemes play an important role in many areas. Not only as the digital equivalent of a pen and paper signature, but also for their use as building blocks to build more complex protocols. Here we review some other forms of signature schemes.

Group signatures enable unlinkable anonymous authentication, in the same way that digital signatures provide the basis for strong authentication protocols. A group signature scheme allows members of a group to sign messages anonymously on behalf of the group, but when a dispute arises a designated revocation manager can revoke the anonymity of a group member. This is useful in many cases, but especially in scenarios where many entities need to submit authentic messages to a central service, without revealing their identity.

Group signatures were introduced by Chaum and van Heyst [71] in 1991 and have since then been the subject of much research. Most of the proposed schemes have a security proof in the random oracle model [10, 33, 58, 59, 128], but since the random oracle model has been shown not to translate into security in the real world [60] there has been some interest in group signature schemes secure in the standard model. Bellare, Micciancio, and Warinschi proposed a security definition of group signatures and also described a construction using trapdoor permutations [25]. However, in that scheme the group is static and all members must be given key material from the start. Bellare, Shi, and Zhang [26] modified the security model to cover dynamic groups, and also split the group manager into two roles; one that could enroll members into the group, and one that could open a signature and reveal the identity of the signer. However, this construction was not efficient. Ateniese, Camenisch, Hohenberger, and de Medeiros proposed an efficient group signature scheme without random oracles [9]. However, if a group member's key is leaked, all his previous signatures can be identified, and the scheme is therefore not secure in BMW/BSZ models. Boyen and Waters suggested group signatures that were secure against the key exposure problem, and anonymous as long as the adversary did not get

to see any openings of group signatures [42, 43]. However, the public key in their scheme grows logarithmically in the size of the message space.

Ring signatures are in many ways similar to group signatures. They allow any member of a group of users to sign on behalf of the group, but in contrast to group signatures there is no way to revoke the anonymity of an individual signature. Furthermore, any group of users can be chosen as members of the "ring", without any additional setup. Ring signatures were proposed by Rivest, Shamir, and Tauman [167] and were originally designed for leaking secrets in an authenticated way, without revealing exactly which individual leaked the secret. Another application is to use ring signatures to build designated verifier signatures, or deniable signatures. A designated verifier signature only convinces a specific recipient of the authenticity of the message, but cannot be transferred beyond its intended recipient. Designated verifier signatures can be realized with a two member ring signature: The signer and the recipient. The recipient knows that he did not sign the message, so the other party must have, but any third party does not know which of the two parties signed it, since the recipient could have produced the signature himself. Designated verifier signatures were proposed by Jakobsson, Sako, and Impagliazzo [121].

Anonymous Credentials. As mentioned in the introduction, a credential is an attestation of qualification, competence, or authority, issued to some subject. That could be a drivers license, a passport or even money could be viewed as a single-use credential granting you the right to spend up to a certain amount. Digital credentials are the digital equivalent of real world credentials, however much more versatile.

Anonymous credential systems, also called pseudonym systems, are systems consisting of users and organizations. Users are known to organizations by virtual identities, called pseudonyms. Credentials are issued to pseudonyms instead of to the actual users, and users are able to prove properties of credentials issued by one organization, to another organizations where the user is known under a different pseudonym, without revealing any more than the fact that the user has a credential. Possession of a credential can be demonstrated an arbitrary number of times and these demonstrations cannot be linked to each other. It must be practically impossible for users to forge credentials even if they cooperate. Also even if organizations collude, they should not be able to find out anything about a user, in particular different pseudonyms belonging to the same user cannot be linked. This is fundamentally different from the situation where a user receives a single X.509 certificate containing all his credentials.

Anonymous digital credentials were first introduced by Chaum in 1985 [68]. Chaum described the general idea and applications, but did not provide a concrete scheme. Shortly thereafter Chaum and Evertse proposed the first concrete protocol realizing these ideas [69]. However, their solution required a semi-trusted third party.

Later Damgård described a protocol without this trusted third party, but the solution is not efficient enough to be used in practise [79]. Brands described an anonymous credential system that allows users not only to show possession

of credentials from different organizations, but also to show properties of attributes encoded into the credentials, such as the age being over eighteen [46]. While this work gives powerful techniques for a privacy enhancing public key infrastructure, one drawback of this solution is that there can be only one organization granting credentials.

In a series of papers, Camenisch and Lysyanskaya identified a key building block in constructing anonymous credential systems [56–58], which is the CL signature scheme described in Section 2.5. Lysyanskaya noted that a signature scheme with the following two protocols could be used to build an anonymous credential system [140]:

Issue This protocol lets a user obtain a signature on a value without revealing this value to the signer. The user gives the signer a commitment to the message and receives in return a signature on that message, without revealing anything else to the signer.

Prove A zero-knowledge proof of knowledge of a signature on a committed value. The user gives a commitment to a message m to the verifier, and then proves that he knows a signature on the message m.

The CL signature scheme satisfies these properties, and can thus be used to build an anonymous credential system. The general idea is the following: The user U chooses a secret key K which will be his identity, and a credential issued by an organization O is a signature on this secret key. If the variant of the CL scheme that allows signing multiple messages is used, O can encode attributes in this credential, such as the users age. Showing the credential, which is essentially a proof of knowledge of a signature, can then be extended to prove certain properties of the signed message. For example that the age is over eighteen. This can be done by making a commitment to the message using an integer commitment scheme [81], proving that the message in this commitment is the same as the message that was signed and finally use the integer commitment and exact range proofs by Boudot [39] to prove that the message committed to is greater than eighteen.

4.2 Unclonable Group Identification

As we have seen, a large body of the literature studies the problem of group identification, where one wants to verify that a given user is a member of a certain group, or has a certain credential, while ensuring that the user's personal identity is not revealed. In some applications, a dishonest user has an interest in giving away to another person the data that allow him to identify himself as a member of the group, such as passwords and secret keys. The security problems implied by such a scenario have not been given much attention so far in the literature. While some earlier works suggest to discourage this by forcing users to either give away *all* their information, or nothing at all, we are interested in cases where dishonest users in fact have an interest in giving everything away.

Here we study this type of problem. As a motivating example, consider the issue of software protection: It is a well known fact that one of the strongest

motivating factors in getting people to register as software users is if this enables some functionality that cannot be accessed without registration (and payment). This works particularly well if the functionality requires access to the vendor's website, because then reverse engineering the software is not sufficient to get unauthorized access to the functionality. In the case of computer games, for instance, the opportunity to play against others may only be available to registered users, and only through the vendor's website.

Verifying that a user is registered may be done in many different ways. In this section, we are interested in solutions that work under the following constraints:

- An honest user can connect an unlimited number of times using the same private key material.
- We want to protect users' privacy, i.e., honest users have to identify themselves *only* as registered users and do not have to reveal their identities.
- We want to do as much as possible to protect against attacks where a user "clones" himself by handing a copy of his personal data (software, secret key(s), etc.) to another person in order to get the benefits of two registrations while only paying for one.

Note that the cloning attack may be easy or very hard to carry out physically, depending on how the user's personal keys are stored, but it can probably never be considered impossible [4, 5].

Of course, we can only hope to detect cloning if the user and his clone actually connect to the vendor's website. A further trivial observation is that if first the user connects, then leaves the site and then the clone connects, we cannot distinguish this from two connections made by an honest user, since he would be using the same private key material in both cases. An event we *can* hope to detect, however, is if both user and clone connect so that they are on the site simultaneously, since this is exactly what cannot occur if the user is honest. In this case, we not only want to detect the attack, we also want to be able to reveal the identity of the user who cloned himself. Note that, apart from the fact that the above simultaneous scenario is the only one in which we can hope to catch a cloning attack, the scenario is also of practical relevance. For instance, the case of a user who buys one copy of a game and distributes it to all his friends so they can play against each other online, is exactly a case where a number of clones would want to be connected simultaneously.

An *unclonable identification scheme* informally, is an identification scheme where honest users can identify themselves anonymously as members of a group, but where clones of users can be detected and have their identities revealed if they identify themselves simultaneously. First we give a formal definition of this primitive and in our Eurocrypt paper we showed that it can be realized assuming existence of one-way functions, which is clearly a minimal assumption [80]. Here we describe a more efficient implementation based on specific assumptions. This solution is based on a new technique for proving in zero-knowledge, given g^x in

a group of prime order, that x was chosen pseudorandomly from a committed secret key.

Of course, before attempting a construction such as we have sketched, one should verify if existing primitives already allow solving the problem. First, one might consider using an anonymous e-cash scheme [45,70], i.e., some number of electronic coins are issued to each user, and users use them to "pay" for access to the site. This would lead to a functionality that is incomparable to the one we sketched above: Cloning in this case means sharing e-coins with others, and so the cloning attack is exactly double spending and can therefore be detected even if the two spendings do not take place simultaneously. But on the other hand, honest users can only use each coin once, and must therefore either possess a very large secure memory, or come back for more coins throughout the life of the system. This reveals information on how often a user connects, and is also not consistent with our goal, namely a solution where you can join a group once and then identify yourself an unlimited number of times using the same key material.

One may also consider using group signatures [9, 10, 33, 59, 128], and have users identify themselves by signing a message chosen by the verifier (using his current system time, for instance). This achieves anonymity but does not protect against cloning. To do this, one would need the property that if the same user signs the same message twice, this would result in signatures that could be detected as coming from the same user. This does not follow from the standard definition of group signatures, and is actually false for known schemes, since these are probabilistic and produce randomly varying signatures even if the message is fixed. A similar comment applies to identity escrow schemes [129].

However, we do want to point out that after this work was published, Camenisch, Hohenberger, Kohlweiss, Lysyanskaya, and Meyerovich, proposed a more efficient solution to this problem [53]. We will sketch their solution in Section 4.2.5.

4.2.1 Definition

An unclonable identification scheme involves a *Group Manager* $\mathcal{GM}$, a set of *Verifiers* and some number of *Users*. The idea is that after some initialization, there will be several events, where some set of users prove *at the same time* to a verifier $\mathcal{V}$ that they are members of the group managed by $\mathcal{GM}$. Since we want to detect if $\mathcal{V}$ is talking to clones of the same user at the same time, every proof should take as input some string α that represents in some sense the current time or phase of the protocol we are in. However, this does not have to be linked to real time. What is important is that whenever a set of users want to prove themselves, they should agree with $\mathcal{V}$ on a value for α that has not been used before. More precisely, the demands are

- An honest $\mathcal{V}$ must be able to ensure that all users he talks to at a given point prove themselves using the same value of α.
- An honest user should be able to ensure that he never executes Prove with the same value of α more than once.

One solution that works in the case where $\mathcal{V}$ runs a website that users would like to be connected to for some length of time, is as follows: At regular intervals, say every hour, each user who is connected must prove himself using the current date and hour as α, as defined by the verifier's system time. This works if there is sufficient agreement on the time between users and $\mathcal{V}$ and if users remember at which time they last did a proof. But many other solutions are possible. Therefore, we have chosen to separate the way time is defined from the definition as such by assuming that the entire system proceeds in consecutive *phases*, with a unique number assigned to each phase. In each phase, some subset of users decide to prove themselves to some verifier $\mathcal{V}$, and the number assigned to the current phase will be used as the string α. An obvious way of doing this is to have the website publish the value of α which is increased every time a new period starts. Users just have to make sure that they only prove themselves when the published α is greater than the last value they used for the proof.

The system is defined by probabilistic polynomial time algorithms Gen, Detect and two-party protocols Join and Prove. These are used as follows:

- Initially, $\mathcal{GM}$ runs Gen on input 1^τ, to get public key pk and secret key sk. We assume for simplicity that the set of possible pk's outputted by $\mathsf{Gen}(1^\tau)$ can be recognized in polynomial time.
- When a user $\mathcal{U}$ joins the system he runs Join with $\mathcal{GM}$. Common input is pk. Private input to $\mathcal{GM}$ is sk. The protocol outputs to $\mathcal{GM}$ either *reject* or a string id. Output to $\mathcal{U}$ is *reject* or a membership certificate $cert_U$. We assume Join is executed on a secure channel so that no other entity will have access to the data exchanged.
- To prove he is a member of the group, the user $\mathcal{U}$ executes protocol Prove with a verifier $\mathcal{V}$. Common input is the public key pk and the string α assigned to the current phase, $\mathcal{U}$ uses $cert_U$ as private input. At the end of the protocol $\mathcal{V}$ accepts or rejects. Each user executes Prove at most once in every phase.
- Algorithm Detect gets as input a number of transcripts of executions of Prove, done with pk as input in the same phase. It outputs a (possibly empty) list of strings. The intuition is that this algorithm should be able to tell if the result of one or more cloning attacks are among a given set of proofs, and if so, it will output the identities of the involved users.

Definition 4.1 *The algorithms and protocols in a secure unclonable identification scheme must satisfy the following:*

Completeness *Assume $\mathcal{GM}$, $\mathcal{V}$ and user $\mathcal{U}$ are honest. Execution of* Gen*, followed by executions of* Join *and* Prove *always result in $\mathcal{V}$ accepting.*

No Cloning *Consider an honest $\mathcal{GM}$ who executes $(pk, sk) = \mathsf{Gen}(1^\tau)$. Consider any probabilistic polynomial time algorithm $\tilde{\mathcal{U}}$ who plays the following game on input pk: In any phase, it can issue one or more of the following requests:*

1. *Ask that a set of honest users execute* Join *with* $\mathcal{GM}$ *(no data returned to* $\tilde{\mathcal{U}}$*).*
2. *Ask to execute* Join *itself with* $\mathcal{GM}$*.*
3. *Ask that some number of honest users who already joined the group execute* Prove *with* $\tilde{\mathcal{U}}$ *acting as verifier, using* pk *and the current value of* α *as input.*

Finally, $\tilde{\mathcal{U}}$ *executes* Prove *a number of times with a honest verifier* $\mathcal{V}$*, on input* pk *and the current value of* α*.*

Let Extract be a probabilistic polynomial time algorithm which gets as input the complete view of $\tilde{\mathcal{U}}$[1] *and outputs a user identity, for every instance of* Prove *that* $\mathcal{V}$ *accepted in the last step. We demand that the following property holds except with negligible probability:*

All user identities output by Extract are among those that were generated in the conversations between $\tilde{\mathcal{U}}$ *and* $\mathcal{GM}$*. Furthermore, the* Detect *algorithm, when given as input the conversation between* $\tilde{\mathcal{U}}$ *and* $\mathcal{V}$*, will output exactly those user identities that occur more than once in the output of Extract.*

Anonymity *Consider any probabilistic polynomial time algorithm* $\tilde{\mathcal{V}}$*, who will act as both* $\mathcal{GM}$ *and verifier in an attempt to break the anonymity of honest users.* $\tilde{\mathcal{V}}$ *gets* 1^τ *as input and outputs a valid* pk*. It then plays the following game: It interacts with a set of honest users, where in each phase some users execute* Join *and other users execute* Prove *with* $\tilde{\mathcal{V}}$*. Of course, no honest user will attempt to do* Prove *unless he already did* Join *successfully. At some point* $\tilde{\mathcal{V}}$ *stops and outputs a bit, and we let* $p_{real,\tilde{\mathcal{V}}}(k)$ *be the probability that 1 is output.*

Consider a different game where $\tilde{\mathcal{V}}$ *interacts with a simulator* $\mathcal{S}$*. The simulator gets as input for each phase the number of users who want to execute* Join *and the number that want to execute* Prove *in the current phase. These numbers are chosen with the same distribution as in the first game. Let* $p_{sim,\tilde{\mathcal{V}}}(k)$ *be the probability that 1 is output in this case.*

We demand that there exists a probabilistic polynomial time simulator $\mathcal{S}$ *such that for any* $\tilde{\mathcal{V}}$*,* $|p_{real,\tilde{\mathcal{V}}}(k) - p_{sim,\tilde{\mathcal{V}}}(k)|$ *is negligible in* τ*.*

Note that the definition of the *no cloning* property implies that if $\tilde{\mathcal{U}}$ did not execute any Join's, there are no user identities *Extract* can legally output, so we are then in fact demanding that all $\tilde{\mathcal{U}}$'s proofs are rejected except with negligible probability. Thus we do not need a separate soundness condition in the definition demanding that non-members are rejected.

In this definition, we have for simplicity used the usual two-phase structure of identification schemes to define soundness and non-cloning, where first the adversary talks to the honest users and then tries to fool the honest verifier. Thus we do not allow him to interact with an honest prover and and honest

[1] This means that *Extract* can rewind $\tilde{\mathcal{U}}$ to any state that occurred during the game.

verifier simultaneously. However, this is not a serious restriction, as there are several techniques that allow handling even this concurrent case, such as the so called designated verifier proofs [78, 121]. A designated verifier proof allows the prover to prove a statement that only a designated verifier can verify. This can be done for any Σ-protocol by using a trapdoor commitment scheme in the commit phase [121]. While the prover cannot cheat, the verifier can use his trapdoor to simulate a transcript for any statement.

As for the scheduling of the individual protocols in a single phase, we consider two cases: One where in each phase the proofs given to a honest verifier are composed sequentially, and one where the composition may be concurrent, with a scheduling chosen by the adversary. We speak of *sequential* and *concurrent security*, accordingly. On the other hand, we assume that honest users (provers) may interact concurrently with an adversarial verifier.

4.2.2 A Practical Solution

In this section, we present an unclonable group identification scheme, based on two main ingredients: First a technique proposed by Camenisch and Lysyanskaya for digital signatures based on bilinear maps, with protocols for proving knowledge of a signature on a committed value [58]. Second, a new technique for proving that an element in a group is of form g^{ψ} where ψ is a pseudorandom value computed from a committed key. We use the notation by Camenisch and Stadler [59]: Given a public string x, a private witness w and a predicate $pred$,

$$PK\{w : pred(x, w)\}$$

means that we execute a Σ-protocol for the relation $\{(x, w) |\ pred(x, w) = true\}$, that is, a prover convinces a verifier that he knows w such that the predicate on x and w is satisfied. We will also use the following variant:

$$PK(\kappa)\{w : pred(x, w)\}$$

where κ is a bit string. This stands for the following: We execute the underlying Σ-protocol in the normal interactive way, except that the verifier sends as the second message a random string κ, and the challenge the prover has to answer is determined as $H(x, a, \kappa)$, where H is a hash function, modelled as a random oracle, and a is the first message in the original protocol. The point of this construction is that it allows simulation of the protocol without rewinding, due to the "programmability" of the random oracle, and for the same reason it also allows knowledge extraction by standard rewinding. Since we will need the last point for the proof, we cannot just use the Fiat-Shamir heuristic.

4.2.2.1 Proofs of Knowledge with Pseudorandom Exponents

Here we introduce some tools to be used in our construction. To this end, we consider a group $\mathbb{G}_p$ of prime order p. We will assume p is chosen as a safe prime, i.e., $p = 2q+1$ where q is also prime. $\mathbb{G}_q$ will denote the unique subgroup of ${\mathbb{Z}_p}^*$ of order q.

We further consider the case where a prover knows exponents $x_1, ..., x_t \in \mathbb{Z}_p$ such that $\beta = \alpha_1^{x_1} \cdots \alpha_t^{x_t}$ for publicly known $\beta, \alpha_1, ..., \alpha_t \in \mathbb{G}_p$. We want a protocol that allows a prover $\mathcal{P}$, to convince a verifier $\mathcal{V}$, that he knows the x_i's. That is, we want:

$$PK\{(x_1, ..., x_t) : \beta = \alpha_1^{x_1} \cdots \alpha_t^{x_t}\} \tag{4.1}$$

A Σ-protocol for this relation works as follows:

1. $\mathcal{P}$ chooses $r_1, ..., r_t \in \mathbb{Z}_p$ uniformly at random and sends to $\mathcal{V}$ $\chi = \prod_{i=1}^{t} \alpha_i^{r_i}$.
2. $\mathcal{V}$ chooses a random challenge $\epsilon \in \mathbb{Z}_p$.
3. $\mathcal{P}$ responds with $z_i = r_i + \epsilon x_i \bmod p$ for $i = 1..t$. $\mathcal{V}$ checks that $\prod_{i=1}^{t} \alpha_i^{z_i} = \chi \beta^{\epsilon}$.

It is well known that this protocol is indeed a Σ-protocol for the underlying relation [173]. A straightforward variant of the protocol allows us to do the following type of proof:

$$PK\{(x_1, .., x_t, x'_1, ..., x'_t) : \beta = \alpha_1^{x_1} \cdots \alpha_t^{x_t}, \beta' = \alpha'^{x'_1}_1 \cdots \alpha'^{x'_t}_t, x_1 = x'_1\}$$

Basically, we run the original protocol twice in parallel for the two equations. This would normally involve two independent sets of random numbers $r_1, ..., r_t$ and $r'_1, ..., r'_t$. However, to demonstrate that $x_1 = x'_1$ the prover must use $r_1 = r'_1$ and the verifier checks that the responses $z_1, ..., z_t, z'_1, ...z'_t$ satisfy $z_1 = z'_1$. We will use this variant later, but for now we stick to the basic version for simplicity. All techniques we describe here can also be applied to this variant in a straightforward way.

We now consider a change to the protocol where $\mathcal{P}$ chooses the randomness in the first message according to a pseudorandom function $\Psi_K(i, \alpha, b)$, where K is a key committed to by $\mathcal{P}$, α is a public input, i is a number and b is a bit. We will use a variant of the pseudorandom function of Naor and Reingold, based on the DDH assumption in $\mathbb{G}_q$, so that outputs from Ψ are in $\mathbb{G}_q$ [152]. We specify below how the function works and how the key is committed. However, in the previous protocol, the random exponents were chosen in $\mathbb{Z}_p$, whereas the pseudorandom function produces output in the subgroup $\mathbb{G}_q$. To resolve this, we let the exponents be chosen as the difference between two pseudorandom values, which allows us to hit all of $\mathbb{Z}_p$. The modified Σ-protocol then works as follows:

1. $\mathcal{P}$ sets $r_i = \Psi_K(i, \alpha, 0)$, $s_i = \Psi_K(i, \alpha, 1)$ and sends to $\mathcal{V}$ $\chi = \prod_{i=1}^{t} \alpha_i^{r_i - s_i}$.
2. $\mathcal{V}$ chooses a random challenge $\epsilon \in \mathbb{Z}_p$.
3. $\mathcal{P}$ responds with $z_i = r_i - s_i + \epsilon x_i \bmod p$ for $i = 1..t$. $\mathcal{V}$ checks that $\prod_{i=1}^{t} \alpha_i^{z_i} = \chi \beta^{\epsilon}$.

To argue that this is a Σ-protocol for the same relation, we need a result by Perron [162]: Let QR_p be the set of quadratic residues mod p. Then for any $a \in {\mathbb{Z}_p}^*$, the set $a + QR_p$ contains almost as many quadratic residues as non-residues: The difference is at most 1. Since in our case $\mathbb{G}_q = QR_p$, we get the following lemma:

Lemma 4.1 *The distribution of $u_i - v_i \bmod p$ where u_i, v_i are chosen uniformly in $\mathbb{G}_q$, is statistically close to uniform over $\mathbb{Z}_p$.*

This allows us to conclude the following:

Lemma 4.2 *Under the DDH assumption in $\mathbb{G}_q$, the above protocol is a Σ-protocol for the Relation 4.1.*

Proof. Completeness is trivial, and special soundness follows exactly as for the previous standard protocol. For honest verifier zero-knowledge, we argue as follows: To simulate we will choose ϵ and z_i at random in their respective domains and then set $\chi = \beta^{-\epsilon} \prod_{i=1}^{t} \alpha_i^{z_i}$.

Now, assuming K is known only to $\mathcal{P}$, pseudorandomness of Ψ implies that our variant is indistinguishable from a protocol where $\Psi_K(i, \alpha, 0), \Psi_K(i, \alpha, 1)$ are replaced by uniformly random choice u_i, v_i from $\mathbb{G}_q$. This creates a distribution of z_i that is statistically close to the simulated distribution by Lemma 4.1. □

Our goal is now to allow $\mathcal{P}$ to prove that he has followed the specified algorithm for choosing the r_i's and s_i's pseudorandomly. The first step of this is to have $\mathcal{P}$ commit to each individual value under a public key chosen by a third party (which will eventually be the group manager in our case). The public key will be two random elements $\eta, \lambda \in \mathbb{G}_p$, and $\mathcal{P}$ will make commitments $com_i = \eta^{r_i} \lambda^{\omega_i}$ and $com'_i = \eta^{s_i} \lambda^{\omega'_i}$, for $i = 1..t$ and random ω_i, ω'_i. We can now ask $\mathcal{P}$ to prove that he committed to the correct values, that is, execute

$$\begin{aligned} PK\{(r_i, s_i, \omega_i, \omega'_i, i = 1..t) : \chi &= \prod_{i=1}^{t} \alpha^{r_i} (\alpha^{-1})^{s_i}, \\ com_i &= \eta^{r_i} \lambda^{\omega_i}, com'_i = \eta^{s_i} \lambda^{\omega'_i}, i = 1..t\} \end{aligned}$$

The Σ-protocol for this is a standard variant of the one we presented above. The final step is to show that each committed value was chosen according to the pseudorandom function. For this we need to specify in detail how it works. We assume that input strings to Ψ all have length at most τ (where τ can in principle be arbitrary). A key to the function is a number $K \in \mathbb{Z}_q$. Finally, we will need a hash function H that take a string str of length at most τ as input and outputs an element in $\mathbb{G}_q$. We will model this function as a random oracle. The pseudorandom function is now defined as:

$$\Psi_K(str) = H(str)^K \bmod p$$

We note that the function mapping y to $y^K \bmod p$ is a weak pseudorandom function if the DDH assumption holds in $\mathbb{G}_q$, i.e., as long as y is randomly chosen and is not controlled by the adversary, the outputs look random. However, in our case, and assuming the random oracle model, the function is only used on values produced by H, and these are guaranteed to be random, even if the adversary chooses the inputs to H. This gives us the following lemma:

Lemma 4.3 *In the random oracle model, and assuming DDH holds in $\mathbb{G}_q$, $\Psi_K()$ as defined above is a strong pseudorandom function.*

We will assume that the key K is committed to by $\mathcal{P}$ in a somewhat non-standard way which, however, fits nicely with the construction we will see in the following. Concretely, we assume that $d = g^{\gamma^{K\delta^r}} h^u$ is given, for publicly known $g, h \in \mathbb{G}_p$ and $\gamma, \delta \in \mathbb{G}_q$. With this, we can summarize our goal, namely to give a Σ-protocol implementing

$$PK\{(K, r, u, \omega_i, \omega_i', i = 1..t) : d = g^{\gamma^{K\delta^r}} h^u \quad , \\ com_i = \eta^{\Psi_K(i,\alpha,0)} \lambda^{\omega_i} \quad , \quad com_i' = \eta^{\Psi_K(i,\alpha,1)} \lambda^{\omega_i'}, i = 1..t\}$$

For this, it will be be enough to show how $\mathcal{P}$ can prove that a given commitment com satisfies $com = \eta^{\Psi_K(str)} \lambda^{\omega}$ for public str. Since anyone can compute $\psi = H(str)$, our task reduces to:

$$PK\{(K, r, u, \omega) : d = g^{\gamma^{K\delta^r}} h^u,\ com = \eta^{\psi^K} \lambda^{\omega}\} \tag{4.2}$$

A protocol for this follows here:

1. $\mathcal{P}$ chooses $s, w \in \mathbb{Z}_q, \nu, \phi \in \mathbb{Z}_p$ at random. He sends $v_1 = g^{\gamma^s \delta^w} h^{\nu}$ and $v_2 = \eta^{\psi^s} \lambda^{\phi}$ to $\mathcal{V}$.

2. $\mathcal{V}$ selects a random bit c.

3. $\mathcal{P}$ sends $z_1 = s - cK \bmod q$, $z_2 = w - cr \bmod q$, $z_3 = \nu - cu\gamma^{s-K}\delta^{w-r} \bmod p$, and $z_4 = \phi - c\omega\psi^{s-K} \bmod p$.

   $\mathcal{V}$ checks as follows: if $c = 0$, that $g^{\gamma^{z_1}\delta^{z_2}} h^{z_3} = v_1$ and $\eta^{\psi^{z_1}} \lambda^{z_4} = v_2$. If $c = 1$, that $d^{\gamma^{z_1}\delta^{z_2}} h^{z_3} = v_1$ and $com^{\psi^{z_1}} \lambda^{z_4} = v_2$.

Since this protocol only works with a single bit challenge, we need to repeat it an appropriate number of times to have a sufficiently small soundness error.

Lemma 4.4 *The above is a Σ-protocol for Relation 4.2*

Proof. Completeness follows by inspection of the protocol. Special soundness: If for given v_1, v_2, the prover can send satisfactory answers z_1, z_2, z_3, z_4 to $c = 0$ and z_1', z_2', z_3', z_4' to $c = 1$, we have by the checks carried out by $\mathcal{V}$ that $g^{\gamma^{z_1}\delta^{z_2}} h^{z_3} = v_1$, $\eta^{\psi^{z_1}} \lambda^{z_4} = v_2$. $d^{\gamma^{z_1'}\delta^{z_2'}} h^{z_3'} = v_1$ and $com^{\psi^{z_1'}} \lambda^{z_4'} = v_2$. Combining these equations imply that $com = \eta^{\Psi^{z_1 - z_1'}} \lambda^{(z_4 - z_4')\psi^{-z_1'}}$ and $d =$

$\alpha^{\gamma^{z_1-z_1'}\delta^{z_2-z_2'}}h^{(z_3-z_3')\gamma^{-z_1'}\delta^{-z_2'}}$, i.e., a, d are of the required form. Finally, honest verifier zero knowledge is argued by the following simulator: Choose z_1, z_2 at random in $\mathbb{Z}_q$, z_3, z_4 at random in $\mathbb{Z}_p$ and c as a random bit. If $c = 0$, set $v_1 = g^{\gamma^{z_1}\delta^{z_2}}h^{z_3}$ and $v_2 = \eta^{\psi^{z_1}}\lambda^{z_4}$. If $c = 1$, set $v_1 = d^{\gamma^{z_1}\delta^{z_2}}h^{z_3}$ and $v_2 = com^{\psi^{z_1}}\lambda^{z_4}$. □

4.2.2.2 The Scheme

We first explain the intuition behind the solution: When joining the group, user $\mathcal{U}$ will make a commitment c_U to a secret key K and will obtain $\mathcal{GM}$'s signature σ_U on the commitment under the signature scheme A by Camenisch and Lysyanskaya [58]. He then proves that he is a member of the group by proving that he knows a valid signature σ_U on some message c_U (as well as proving knowledge of this value), without revealing either value. Moreover when giving this proof he uses some pseudorandom values that he obtains by applying a pseudorandom function to the current α value, using K as key. He also proves that he has done exactly this. This will force a clone of the user to use the same K if he gives a proof for the same α, by security of the commitment and signature schemes.

The proof given is actually a Σ-protocol, so the transcripts of proofs given by user and clone are of form (a, e, z) and (a', e', z'). But when all inputs are the same in the two cases, we must have $a = a'$. Furthermore, $e \neq e'$ with overwhelming probability, so if both proofs are accepted, special soundness of the protocol means that one can easily compute the prover's secret, which will immediately identify the user in question.

Gen Let $\mathcal{GM}$ take a security parameter τ and output a group $\mathbb{G} = \langle g \rangle$ of prime order $p = \Theta(2^\tau)$ where $p = 2q + 1$ and q is a prime. Let $\mathbb{G}_q$ denote the unique subgroup of ${\mathbb{Z}_p}^*$ of order q. Let γ, δ be random generators of $\mathbb{G}_q$. Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be an efficiently computable bilinear map, and η, λ be random generators of $\mathbb{G}_T$.

To set up the signature scheme, $\mathcal{GM}$ chooses the following values at random: $x \in \mathbb{Z}_p$, $y \in \mathbb{Z}_p$ and sets $X = g^x$, $Y = g^y$. The secret key for the signature scheme is $sk = (x, y)$ and the public key is $pk = (\mathbb{G}, \mathbb{G}_T, g, \mathbf{e}, X, Y, \eta, \lambda)$.

Join The user $\mathcal{U}$ chooses at random $r_U \in \mathbb{Z}_q$ and a key $K \in \mathbb{Z}_q$. $\mathcal{U}$ makes a commitment $c_U = \gamma^K \delta^{r_U} \bmod p$ to K and sends it to $\mathcal{GM}$. Note that it is not necessary to have $\mathcal{U}$ prove that he can open c_U, since later in the Prove protocol, he must implicitly show he can open it to have the proof accepted.

$\mathcal{GM}$ verifies that $\mathcal{U}$ is allowed to join the group and if so, he computes a signature $\sigma = (a, b, c)$ on c_U where a is chosen at random in $\mathbb{G}$, $b = a^y$, $c = a^{x + c_U xy}$ and sends it to $\mathcal{U}$. $\mathcal{GM}$ considers c_U as the user's *id* in the following, whereas (K, r_U, a, b, c) serves as the membership certificate.

Prove Recall that the string α is common input to $\mathcal{U}$ and $\mathcal{V}$. First $\mathcal{U}$ blinds his signature σ by choosing at random $\mu, r' \in \mathbb{Z}_p$ and computing $\widetilde{\sigma} = \left(\widetilde{a}, \widetilde{b}, \hat{c}\right)$ where $\widetilde{a} = a^{r'}$, $\widetilde{b} = b^{r'}$, $\hat{c} = (c^{r'})^{\mu}$. Then $\mathcal{U}$ makes a commitment $C_U = \eta^{c_U} \lambda^{s_U}$ for random s_U and sends $\widetilde{\sigma}$ and C_U to $\mathcal{V}$. Both compute

$$v_x = e\left(X, \widetilde{a}\right), \quad v_{xy} = e\left(X, \widetilde{b}\right), \quad v_s = e\left(g, \hat{c}\right)$$

$\mathcal{V}$ chooses a τ-bit string κ at random, and $\mathcal{U}$ proves knowledge of a signature on c_U to $\mathcal{V}$ by giving the following proof:

$$PK(\kappa)\{(c_U, \rho, s_U) : C_U = \eta^{c_U} \lambda^{s_U}, {v_s}^{\rho} = v_x v_{xy}^{c_U}\} \tag{4.3}$$

Here, as ρ, the honest $\mathcal{U}$ uses $\rho = \mu^{-1} \bmod p$. $\mathcal{V}$ will accept if this proof is correct and it holds that:

$$e\left(\widetilde{a}, Y\right) = e\left(g, \widetilde{b}\right)$$

When applying the Σ-protocol from Lemma 4.2 to Relation 4.3, the part relating to $C_U = \eta^{c_U} \lambda^{s_U}$ will have a first message of the form

$$\chi = v_{xy}^{r_1 - s_1} v_s^{r_2 - s_2},$$

Where the r_i's and s_i's are generated pseudorandomly as

$$r_1 = \Psi_K(1, \alpha, 0), r_2 = \Psi_K(2, \alpha, 0), s_1 = \Psi_K(1, \alpha, 1), s_2 = \Psi_K(2, \alpha, 1)$$

$\mathcal{U}$ must prove this fact, so he sends commitments

$$com_1 = \eta^{r_1} \lambda^{\omega_1}, com_2 = \eta^{r_2} \lambda^{\omega_2}, com_1' = \eta^{s_1} \lambda^{\omega_1'}, com_2' = \eta^{s_2} \lambda^{\omega_2'},$$

and proves them correct with respect to χ and uses the protocol from Lemma 4.4 to show that each commitment contains a pseudorandom value of correct form.

Detect Look at all proofs given in a phase and find all places where two conversations include first messages χ, χ' where $\chi = \chi'$. If the two challenge values involved in these two conversations are different, use the special soundness property to extract a witness for the proof in question – this will be a pair of form (c_U, ρ). Output all c_U's found this way.

Theorem 4.1 *Assuming security of the signature scheme and the DDH assumption in $\mathbb{G}_q$, the scheme described above is a secure unclonable identification scheme in the random oracle model, with sequential security. The* Join *and* Prove *protocols are constant-round, and have communication complexity $O(k)$ bits, respectively $O(k^2)$ bits.*

Proofsketch. Completeness follows by inspection of the protocols. To achieve the no cloning property, the required *Extract* algorithm will use standard rewinding to extract witnesses for all proofs given. By a standard argument, this will succeed for all proofs that were accepted by the verifier, with overwhelming probability. Soundness of the proofs means we will extract a set of user id's and corresponding signatures, so security of the signature scheme implies that this forms a subset of the user id's defined in previous Join protocols. Now, soundness of the proofs from Lemma 4.4 and the binding property of the commitment schemes defined by $(\gamma, \delta), (\eta, \lambda)$ imply that the adversary must have used the key involved correctly and consistently, and hence the value of χ will be identical in all instances of Subproof 4.3, where the same key was used. This allows Detect to recover the required information. As for anonymity, note that all subproofs except the one from Subproof 4.3 can be replaced by (perfect) simulations without changing the view of the adversary. After this change, the key K is only used to call the pseudorandom function, and no other information on K is present, since the commitment c_U hides K perfectly. We can therefore use Lemma 4.2 to conclude that also instances of subproofs from Equation 4.3 can be replaced by simulations without this being detectable by the adversary. □

Efficiency We only consider the efficiency of Prove since the resources required by Setup and Join are negligible compared to the resources required by Prove. Note that all proofs given during Prove can be done simultaneously, using the same challenge in all Σ-protocols.

If we set $k = 512$ (a reasonable value for security equivalent to 1024 bit RSA) and $r = 16$, where r is the number of times the protocol from Lemma 4.4 is repeated, Prove requires more than 1000 exponentiations, divided more or less evenly between $\mathcal{U}$ and $\mathcal{V}$). The exact numbers depend on various optimizations one can perform by running some of the proofs in parallel, but it seems unlikely to be able to get below 1000 exponentiations. Even with the choice of $k = 512$, we would need to transmit more than 100.000 bytes of data between $\mathcal{U}$ and $\mathcal{V}$. Hence this is more a proof of concept, than a practical implementation.

4.2.3 On Concurrent Security

All the proofs given by honest users can be simulated without rewinding. Hence, the only problem in obtaining concurrent security lies in the *Extract* algorithm that is required for the no cloning property, and which requires rewinding in both solutions.

To avoid this, we can use, at a small efficiency cost, the technique by Fischlin [94], which shows how to transform any Σ-protocol in the random oracle model into a new one for which there is an on-line extractor, i.e., one can extract the secret witness from a successful prover without rewinding.

Using this transformation on the Σ-protocols underlying our Prove-protocol immediately gives a concurrently secure solution.

4.2.4 On Membership Revocation and Framing

After discovering the identity of a dishonest user, the group manager needs to act. In some applications it may be sufficient to take some appropriate, say, legal action against the user in question. But it may also be necessary to remove the user out from the group by ensuring that the value c_U can never be used again.

Since the value c_U is unconditionally hidden in the Prove protocol, nothing in the above systems prevents a dishonest user from proving membership of the group again at a later point in time. To allow for revocation of memberships, we can extend the protocol with a *dynamic accumulator* as described in [57]. An *accumulator scheme* [20,27] is an algorithm that allows one to hash large set of values into a short value, called the *accumulator* such that there is a witness that a given input is in the accumulator. A *dynamic accumulator* allows one to efficiently add and remove values from the accumulator. It can be used in the following way.

When the user joins the group and sends c_U, the group manager adds c_U to the accumulator. To prove membership of the group, the user is now required, in addition to the protocol we already have, to prove that the value c_U is in the accumulator. We will omit the details of how this is done, they can be found in [57]. The solutions needs that c_U is committed to, but this is already done in our protocol.

When the identity of a dishonest user is discovered, the group manager removes c_U from the accumulator, which prevents the user or any clones of the user from proving membership of the group, as long as the verifier is aware of the new accumulator value.

An aspect we have not been concerned with in this work is whether the group manager can *frame* an honest user, that is, create on his own a protocol transcript where the user seems to have cloned himself. We believe framing is not possible, since the group manager does not know the user's secret key K. The part of the Prove protocol where the user proves knowledge of K can only be simulated without knowing K if one can control the outputs of the random oracle. While we use this to show zero-knowledge in the theoretical analysis, such control is not available to anyone in real life.

4.2.5 A More Efficient Solution

Inspired by our work, Camenisch et al. published a solution solving the same problem much more efficiently [53].

We argued that e-cash was not a feasible solution since by nature a coin can only be spent once. Hence it would require the user coming back to retrieve more coins, which in turn would reveal information about how often he uses the service. However, as Camenisch et al. showed, one can still use the idea from e-cash, and especially compact e-cash, to solve this problem. In compact e-cash schemes, users are issued cash in form of dispensers, that can dispense a limited number of coins. However by allowing a dispenser to dispense an unlimited amount of coins in general, but only allowing it to dispense a limited number

of coins per time period, techniques from compact e-cash can in fact be used to obtain an unclonable identification scheme.

Camenisch et al. furthermore extends their protocol to allow up to n authentications per time period, arguing that it might not always be possible for a user to keep exact track of the time when the last authentication was performed.

A dispenser consists of the seed s for a pseudorandom function Ψ, the users secret key sk and the issuers signature on those values. When the user wants to show coin number i, he generates two values $S = \Psi_s(0, \sigma, i)$ and $E = pk \cdot \Psi_s(1, \sigma, i)^R$, where σ is the time period and R is a random value chosen by the verifier. Furthermore the user proves in zero-knowledge, that S and E corresponds to a valid dispenser for the given time period. If a user shows more than n coins during a time period, two of the coins must use the same value of S. The verifier can then easily compute pk from the two different values of E.

Their solution requires only 13 multi-base exponentiations to perform one authentication when $n = 1$ and 35 multi-base exponentiations when n is greater than one, which is orders of magnitude more efficient than our protocol.

Chapter 5

Usability and Authentication

In this chapter we look at scenarios where the user must be authenticated towards some system.

First we give an introduction to some security issues relevant for the rest of this chapter in Section 5.1. In Section 5.2 we give a brief overview of our proposed solution for a login system, to improve usability of pervasive computing in a hospital. This is based on a paper presented at the UbiComp 2003 conference [18] and has later been implemented as part of the Activity Based Computing framework [156]. Finally, in Section 5.3 we look at the problem of theft of pervasive computing devices, and propose a method for reducing that risk. This is based on a paper presented at the AINA 2007 conference as well as some additional work based on this.

5.1 Introduction

In the previous chapters we looked at ways of providing security in various scenarios, with focus on which cryptographic techniques to use. However, in any real world system there is one threat to any system that we simply can't ignore, and that is the legitimate users of the system. This issue becomes more important when we work in the realm of pervasive computing, as we will often find many computing devices in the homes of average users with little or no knowledge about security. To make the issue even more complicated, many of the devices in a user's home will be small wireless devices, with limited computational resources, and a different form of user interface (if any) than what the user is acustomed to. In this chapter we present our work in two different areas, with different security requirements. In the first scenario we have many users who have to log in to computer systems multiple times during the day in a hospital environment. Here the goal is ease of use and secure authentication of the users. In the second scenario we want to secure a dynamic network of devices in a user's home. Being dynamic means that the user must be able to add and remove devices from the network at will, which means that ease of use is a key requirements. Adding devices to a network nesecerily means that the devices must somehow exchange cryptographic key material. First we develop a security policy for this setting and then apply this to key distribution

and theft protection in a home network.

Usability. There is a saying that computers are very easy to secure. Just turn them off, lock them in a vault and throw away the key. That will however make computers quite unusable. A more everyday example, is the problem with passwords. Not using passwords at all, makes it very easy for users to use a system, but not very secure against unauthorized access. Requiring long passwords with a combination of letters and symbols, that must be changed every week, is probably more secure, but will significantly hinder users in their daily use of the system.

It has been argued for a long time that usability and security must go hand in hand, yet it seems that in more or less every system today, the more security features a given system has, the more cumbersome it is to use, but if security features are optional (such as e-mail encryption) they will not be used [191]. There seem to be three different views on how to build usable security into systems. The first is to build systems that perform security related functions without any user intervention. Unfortunately this can be very difficult to realize in practise for several reasons. One being that users might act in ways that undermine the protection put in place, if they are unaware of the security issues related to the work they are performing. Another reason might be that security involves only allowing legitimate users to perform some action or access some resource, but this requires configuration of a system to determine who the legitimate users are, and it requires a way for the users to authenticate themselves. Hence there is some security overhead that seems hard to get rid off. The second idea is to build secure systems using metaphors that the user already knows. The ”lock and key” metaphor is well known, but the problem is that many of the security and privacy issues users face today, do not have a corresponding metaphor that the user is used to handle in the real world. The final solution is to educate users to use the security features a product provides. This has over time proven to be quite unsuccessful. There is simply too much to learn for the average user.

Security and usability, are features that must be designed into a system from the beginning, but they require quite different skills to get right, which might be one of the reasons it seems to hard to combine them [158, 177].

Based on previous work, Pagter and Petersen [157] highlights the following overall design principles for usable security:

- Design for a specific context
- Establish coherence between ”normal” actions and security actions
- Make security states visible
- Implicit infer security actions from user actions
- Use explicit security actions when users need to act in response to significant security risks

In Section 5.2 and 5.3 we will see some of these design principles used.

User Authentication. When using a personal computer, typing in username and password is straightforward, but still it poses substantial usability problems in some work environments, like hospitals [19, 172]. Looking at pervasive computing, these usability problems are increasing because the user is using many computers. Imagine that a user would need to type in username and password on all pervasively available computers before he could start using them. Clearly, if the pattern of login and logout is not considered a usability problem today, it will most certainly become one in the years to come.

One solution is what we have termed *proximity-based login* (see also [76]), which allows users to be authenticated on a device simply by approaching it physically. The idea of enabling users to access a computer by simply walking up to it has a long history in pervasive computing research. This idea of proximity-based login can be traced back to the pioneering work on the Active Badge System, which could be used to teleport an X Window session to a display located in front of the user [28, 187]. Similarly, the AT&T Active Bat system [188] was used to create "Follow-me Applications" where the user-interfaces of the user's application can follow the user as he moves around. The application is shown on the display that is deemed to be in front of the user as established via the user's Active Bat [111]. The idea has later been adopted by the Microsoft EasyLiving project [49]. In other pervasive computing environments different types of physical tokens are used as user identification. In the BlueBoard project at IBM, a HID brand reader is used [168, 169]. In the AwareHome at Georgia Tech RFID tags provide identity of individuals near commonly used monitors [151], and at FX PAL the Personal Interaction Points (PIPs) System uses RFID cards that stores the users identification as well as passwords (the latter in encrypted form) [184]. Ensure Technologies' XyLoc system [88] uses short-range radio communication between a personal token and a PC to establish the proximity of a user and to unlock the PC by pressing a button on the token, not unlike the ones used to remotely unlock cars.

Key Exchange. Exchanging cryptographic keys are an important issue in almost any system that relies on cryptography to implement some form of security. Many problems such as keeping data confidential or only allowing authorized entities to communicate, can be solved by well known (and not so well known) cryptographic techniques, but in order to use these techniques we need some key material. Since many small devices are not capable of performing the computations needed for public key cryptography, sensor networks and other scenarios where many devices need to communicate securely, often rely on symmetric cryptography, but then key distribution becomes problematic. Many devices do not have a user interface for entering passwords, they might be placed in hard to reach locations, or it might simply be too cumbersome for the user to manually distribute key material. Furthermore, because of the limited resources available on these devices, it can be difficult to authenticate them towards each other, to make sure that cryptographic keys are only exchanged between authorized devices.

In cryptography one usually assumes an adversarial model where the ad-

versary can monitor and modify all communication in the network. However, such an attacker rarely exist in the real world where devices spread over a relatively small area (such as a house) need to agree on key material. For this reason some alternative adversarial models have been considered. For example, Anderson, Chan, and Perrig propose a scheme for wireless sensor networks, where a device starts by broadcasting a symmetric key with low power in the clear [7]. If it does not receive a reply from another device, it increases the transmit power and tries again. They call this whisper-mode and is supposed to limit the range from where the key can be eavesdropped. When all devices have agreed on a key with one of their neighbors, they start mixing their keys in the following way. Assume we have three nodes A, B, and C and three keys K_{AB}, K_{AC} and K_{BC} which are used to protect communication between respectively AB, AC and BC. If A wants a stronger key between himself and B he asks B and C to negotiate a new key based on the previous key between A and B in such a way that if K_{AB} was secure before, then the new key K'_{AB} is also secure, but if K_{AB} was compromised, then K'_{AB} is still secure if K_{AC} and K_{BC} are secure. If this is done at regular intervals by all devices, an adversary has to be able to monitor all traffic to keep track of the keys. If he misses just one key-exchange message, then after a while he will no longer have access to any keys in the network.

However, in commercial applications the solution applied is usually much more straightforward. In the ZigBee standard, each network has a unique symmetric encryption key that is broadcast in the clear to new devices when they join the network. For example, the user presses a button on a master device, and for the next 30 seconds, a new device is allowed to enter the network.

No matter now the implementation is done, such an initial key exchange is often modelled after the resurrecting duckling principle proposed by Anderson and Stajano [179], where a device accepts a key from the first device it receives data from when it is turned on. However, this naive implementation does not protect against an adversary who sets up a device with a more powerful signal, so the new device will recognize it as its mother duck (for example, how many have tried using the neighbors WiFi access point by mistake?). We will describe the resurrecting duckling principle in more details later. One solution against an adversary monitoring the initial key exchange, or an adversary pretending to be the mother duck, is to use authentic channels for the initial key exchange [16]. For example by requiring physical contact between devices.

Often people argue that security, or rather cryptography, is unnecessary, based on the limited range of the wireless technology used. The problem is that the effective range of wireless technology depends on power levels, etc. dictated by the standard, but an adversary is not forced to follow the standard. WiFi has an advertised operational range of around 100 meters, yet it has been deployed over a distance of 382 km [127]. Bluetooth has a range of around 10 meters, but has been eavesdropped from over 1000 meters away [74].

Another problem arises when a single encryption key is used by all devices, since compromising just one device leaves the entire network vulnerable to eavesdropping and message forgery. Also this technique is not suited in scenarios where you need to authenticate messages from individual devices.

Theft Protection. Rarely is old-fashioned theft of the physical devices in pervasive computing considered, despite of the fact that recent studies indicate that this one of the major security issues for pervasive computing. Often devices are not stolen, but rather forgotten in meeting rooms, taxis, etc. [163]. Another security issue in pervasive computing is related to high-end entertainment systems (TV sets, HiFi equipment, gaming consoles, etc.) which are often the target of systematic burglary.

Anti-theft policies and solutions are not widespread in the literature, maybe because designers are afraid to disclose the inner workings of their mechanisms. However, security-by-obscurity is rarely a good solution in the long run. One recent contribution in the literature by Droz, Gülcü, and Haas, describe a system relying on credits and blacklists, with each protected device periodically requesting new credit in order to continue operating [86]. Another source on anti-theft is Ross Anderson's book on security engineering [6] which contain the description of various solutions (not only for anti-theft) and their advantages and disadvantages.

5.2 Context-Aware User Authentication

One thing alle the solutions based on *proximity-based login* have in common is the lack of proper security mechanisms that can effectively ensure a secure user authentication. In case of theft of a token, or by recording and replaying the communication between the token and the reader, an adversary can access the system and impersonate the legitimate user. The Smart Badge platform [178] uses some improvements to reduce the problem of stolen tokens. The method is to use a badge that can detect when it is no longer being carried. The link between the badge and a particular user can then be removed, and the badge can no longer be used for authentication purposes. It is however difficult to judge from the paper exactly how such a smart badge can sense whether is is being carried by its legitimate user. The Zero-Interaction Authentication method of [77] takes an approach similar to the XyLoc system. A token is used to gain access to a laptop with an encrypted file-system. To verify the user, he is required to enter a PIN code on the token before he can start using it.

There is often an inherent tradeoff between usability and security. User authentication mechanisms tend to be either secure, but less usable, or very usable, but less secure. It is our aim to try and combine the two standpoints and suggest a *context-aware user authentication mechanism* that is very usable as well as sufficiently secure for use in settings, where security matters, e.g. a hospital environment. Traditionally a user authentication mechanism is considered secure if it is a combination of something the user *has* (e.g. a smartcard), something the user *knows* (e.g. a password), or something the user *is* (i.e. a fysiological trait). Our design of a user authentication mechanism is based on supplementing well-known user authentication mechanisms with knowledge about the context and location of the user. In line with Denning [83] we thus suggest location-based authentication and introduce *location* as a fourth element in a user authentication mechanism.

Troubles with Login. In a study of the use of Electronic Patient Records (EPR) at a large metropolitan hospital, we observed a number of usability problems associated with user authentication [19]. The EPR was accessed through PCs distributed within the hospital, and it had a traditional login system with usernames and passwords. Thus, whenever a clinician should access patient information he had to log in and out on different PCs. Due to the way the PCs were deployed and the nature of the work in hospitals, it was not uncommon that a nurse, for example, would log in 30 times a day. Because this was a highly cumbersome thing to do in a hectic environment, workarounds were established. For example, users would avoid logging out, enabling them to return to the PC without logging in later, passwords were shared among users and made very easy to remember ('1234' was the most used password at the hospital), and users would often hand over user sessions to one another, without proper logout and login. Hence, what was designed to be a secure system (with traditional username and password user authentication) was suddenly turned into a highly insecure system, because of obvious usability problems.

Activity-Based Computing. Even though this EPR system can in no way be termed as pervasive technology, our study of its use has highlighted how essential user authentication is to the design of pervasive computer support for medical work in hospitals. In our second line of research we actively design and develop pervasive computing technologies for hospital work. A central component in this effort is a basic runtime infrastructure, which supports Activity-Based Computing (ABC) [76]. The basic idea of activity-based computing is to represent a user's (work) activity as a heterogeneous collection of computational services, and make such activities available on various stationary and mobile computing equipment in a hospital. Clinicians can initiate a set of activities, and access these on various devices in the hospital. For example, a nurse can use the computer in the medicine room to get some medicine, and later when giving this medicine to the patient she can restore the patient and medicine data on the display in the hospital bed. We have built prototypes of wall-size displays, displays embedded in tables, built-in computers in hospital beds, and we are using various mobile equipment like TabletPCs and PDAs. Thus, activity-based computing allows users to carry with them, and restore, their work on heterogeneous devices in a pervasive computing environment. Central to this is clearly that users need to be authenticated on every device they want to use, and easy login is hence a core challenge in the concept of activity-based computing.

Our design is based on participatory design sessions and workshops with a wide range of clinicians, including physicians, radiologists, surgeons, and different types of specialized nurses. All in all 12 such workshops were conducted, each lasting 4-6 hours having 6-10 participants each. Various aspects of designing support for clinical work were discussed, including the login mechanisms. Several user authentication mechanisms were designed, implemented, and evaluated in these workshops.

Requirements. Based on existing research within UbiComp, our studies of medical work, and our experimental design effort with end-users, we can list the following requirements for a user authentication mechanism in a pervasive computing environment.

Proximity-Based Work in a hospital is characterized by busy people who are constantly moving around, and are engaged in numerous activities in parallel. Easy and fast login was thus deemed a fundamental prerequisite for the success of a distributed, pervasive computing infrastructure, embedded in walls, floors, tables, beds, etc. The usability goal in our workshops reached a point where the user should do nothing to log in. He should simply just use the computer, and the computer would know who he was.

Secure Clinical computer systems store and handle sensitive, personal health data for many patients. It is therefore of utmost importance that these systems are protected from unauthorized access. Hence, pervasive computer systems in a healthcare environment require secure user authentication.

Active Gesture We experimented with a login mechanism that would automatically transfer a user's on-going session to a display near him, much like the 'Follow-me' application using the Bat system [112]. This, however, turned out to be a less useful design. The problem was that often a clinician would enter a room, where numerous computing and display devices would be available. For example in a radiology conference room, there would be several wall-based displays, a wide range of desktop computers, and an interactive table where images can be displayed and manipulated. It was unclear from monitoring the location of the user, which of such displays he would like to use, or whether he wanted to use a computer at all. Therefore the authentication mechanism must be based on an active gesture near the display or devices that the user wants to use.

Support for Logout During our experiments we discovered that the process of logging out a user is equally important. Clinicians would often have to hurry on, and would simply walk (or run) away from an ongoing session. In this case, automatic logout was deemed important.

One might be surprised to find that privacy is nowhere to be found in the requirements, but since the system will only be deployed inside a hospital environment, there were no privacy concerns from the people participating in the workshops. Furthermore one goal of the system already being developed is to be able to determine the location of people inside the hospital, so e.g. the nearest physician can be quickly located in case of an emergency.

5.2.1 Our Design

There are three key principles in our design of a context-aware user authentication mechanism. First, it uses a physical token for active gesturing and as the

cryptographic basis for authentication. Second, it uses a context-awareness system to verify the location of the user, and to log out the user when he leaves the computer(s) in a certain place. Third, it contains fall-back mechanisms, so that if either of the two components in the system falls out, the user authentication mechanism switches to other mechanisms. If the context-awareness infrastructure is unreachable for any reason, the user is requested to enter his password when trying to log in. If the token cannot be accessed for any reason, the user is requested to enter both his username and password, as usual. Hence, in case of system failure, security is not completely compromised, and the system is still usable. We shall return to a more detailed security analysis below.

Our current design uses smart card technology and in the rest of the paper we will present this design based on a smart card as the physical token. Furthermore, we shall only consider the part of the authentication protocol that involves the smart card. Standard authentication using username and password is not further discussed. The two basic components in the context-aware user authentication mechanism are (1) a secure user authentication protocol between a computer and a smart card, and (2) a distributed computing context-awareness infrastructure.

5.2.1.1 Authentication Protocol

The authentication protocol is running on a smart card. Each computer is equipped with a card reader and the protocol is executed every time a user inserts his card into the reader on the client. In order to use the card for authentication, the following information is stored on the card:

- An *id* for the user the card belongs to.
- The user's *password*.
- The user's secret key (K_S) and public key (K_P).

When the card is issued, a program for authentication is stored on the card, and initialized with the user's *password* and the ID of the user. When the applet is initialized, the card creates an RSA key-pair. The secret key (K_S) is stored on the card and the public key (K_P) is stored in a central server along with the ID of the user. The authentication protocol is illustrated in figure 5.1 and consists of the following steps:

1. The client receives notification that user P is in the room (optional).
2. The user places his smart card in the card reader.
3. The client requests the ID from the smart card.
4. The client looks up the person in the context server based on the ID from the card.
5. There are two distinct cases based on the probability that the user is in the same place as the client.
   Case A: The probability is greater that a certain threshold.

- The smart card is asked to verify that it holds the user's secret key, K_S.

Case B: The location of the user is not sufficiently sure.

- The computer asks the user to enter his *password*.
- The smart card accepts or rejects the user based on the password.

6. The user is either denied or allowed access.

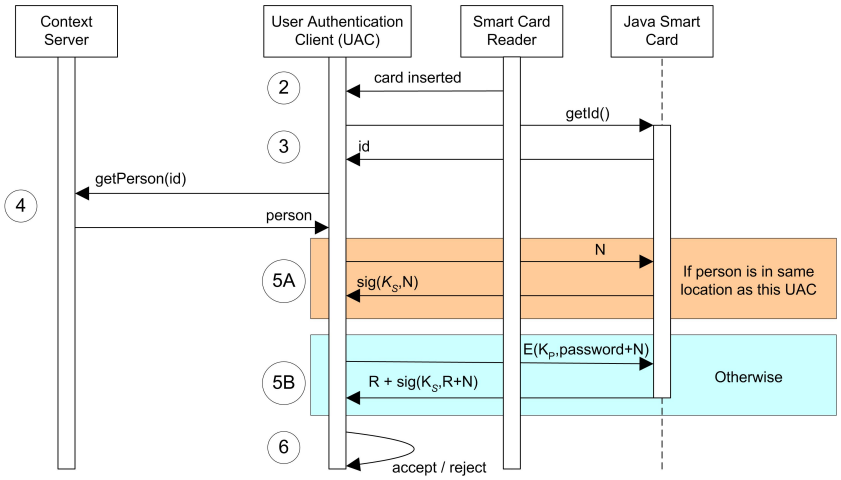


Figure 5.1: Interaction Diagrams for the Authentication Protocol. Case A – The person is in the same location as the User Authentication Client (UAC). Case B – The person is not in the same place, and the user is requested to enter his or her password.

In case A, where the user is known to be in the room, the client verifies that the smart card knows the user's secret (private) key K_S by generating a random 20 byte "nonce", N, and sends it to the card. The card then sends back the signature under the private key, $sig\,(K_S, N)$, of N, and the client uses the corresponding public key, K_P, to verify that the signature is correct.

In case B, the user is not known to be in the room. The client asks for the user's password, concatenates it with a 20 byte nonce N, encrypts it under the user's public key, $E\,(K_P, password + N)$, and sends it to the card. Since the card knows the secret key, it can decrypt this message. It then compares the received password with the one stored on the card. If they match the card returns a byte R and a signature on R concatenated with N, $R+sig\,(K_S, R + N)$, where R equal to 1 is accept and 0 reject. The client uses the public key, K_P, to verify the signature.

5.2.1.2 Infrastructure

The system architecture for the context-awareness infrastructure is illustrated in figure 5.2, and consists of the following main components:

Context Monitors A range of hardware and/or context data specific processes, which register changes in the environment. Examples of context monitors are location monitors based on monitoring RFID tags attached to items in the environment, or WiFi monitors that try to locate WiFi-based equipment. Other monitors might gather information about temperature, planned activities in users' personal calendars, or try to identify people in a room based on their voices.

Context Server The context server contains a simple data structure that stores information about *entities* in the environment. Entities are usually people, places, or things, but this structure is extensible and all kinds of context data can be stored by implementing some simple interfaces.

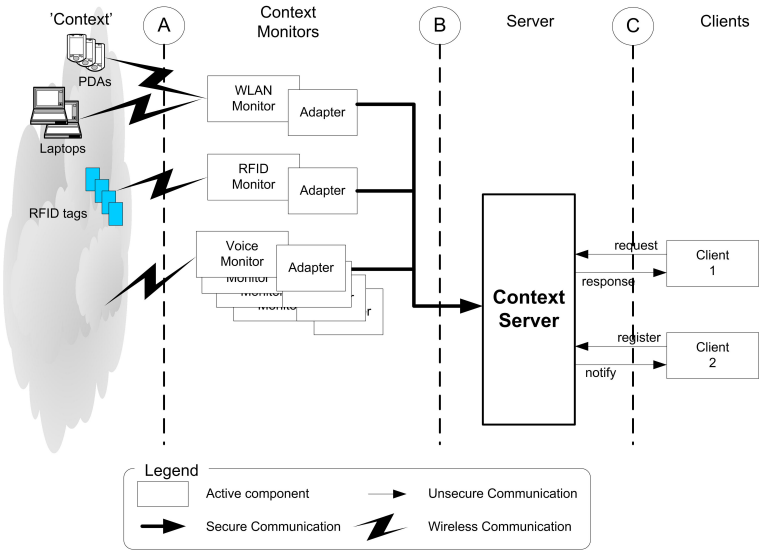


Figure 5.2: System architecture for the context-Awareness Infrastructure.

From a client's point of view there are basically two ways to connect to the context server. On one hand, a client can look up an entity and ask for its context information, like location. For example, a client can request a person and ask for the location. On the other hand a client can register itself as a listener to an entity, and it will hereafter receive notifications when changes to this entity take place. For example, a client running on a stationary computer (e.g. a wall-sized display) can register itself as listener for changes to the place in which it is located. Whenever entities enter or leave this place, the client will be notified.

In our authentication protocol, the context server is asked for a person's location. The confidence in the answer from the server can be divided into addressing two questions:

How accurate is the location data? Whenever the context server provides a location of an entity, it estimates the accuracy of this location. Thus, the

context server embeds an *accuracy algorithm*. In step 5 of the protocol, the accuracy of the location estimate is compared against a configurable threshold value.

Do we trust the location data? This is a question of whether we trust the information that is stored in the context server. Because monitors are the only way location data can be altered in the server, this reduces to a question of trusting the context monitors. To prevent non-authorized monitors to access and update the context server we require the monitors to authenticate themselves to the server. Hence, our context-awareness architecture supports secure authentication and access in layer B in figure 5.2 between the monitors and the server. However, there is currently no security in layer A between the tokens and the monitors, because RFID tags do not have processing power to support a cryptographic setup. We shall return to the security consequences of this below.

The communication layer C between the context server and the clients is not secure. Hence, a non-authorized client can get access to the server and read data from it. This might be a problem from a privacy point of view, but from a security point of view clients cannot alter location data in the context server.

5.2.2 Security Analysis

We will now take a look at the security of this system. The goal of the adversary is to authenticate as a legitimate user. We will make the assumption that the adversary has access to all information stored about the user, except information stored on the smart card (the private key and the user's password). We will also assume that communication between the context monitors and the context server is secure and that only legitimate context monitors can access the context server. Finally we assume that the information stored on a smart card cannot be read or changed except through the designed interfaces.

We also assume that the protocol between the smart card and the reader is a secure one way authentication protocol. We have implemented this as a fairly standard protocol from the litterature. The protocol can be shown secure using a technique called BAN logic by Burrows, Abadi, and Needham [52].

5.2.2.1 Passive Attacks

In the passive attack scenario we assume that the adversary can monitor all communication between the smart card and the terminal. In the case where location data is based on a token the user has, the adversary can also monitor all communication between that location token and the context monitor. Using the information he acquires during this phase he will now try to impersonate the legitimate user after having acquired the user's smart card, both the smart card and the location token (if one is used) or none of them. We have identified the following passive attacks:

1. If the adversary acquires the smart card and is able to fake the location of the legitimate user (by stealing the location token or cheating other

devices used to establish the location of the user) he can authenticate as the legitimate user since to all parts of the system he is that user.

2. If he acquires only the smart card he can authenticate as the legitimate user only in the location where the user is actually present.

3. If the user is not present he cannot authenticate as that user even though he has the smart card, unless he also knows the user's password.

5.2.2.2 Active Attacks

Where the passive adversary could only look at messages between the location token and the context monitor and the smart card and the terminal, the active adversary can drop, change, inject or completely replace any of these messages. He can also create his own smart card using the information he obtains. The goal is the same, though: After having done this for as long as he wants he will now try to impersonate the legitimate user. We have identified the following active attacks:

1. The adversary can retransmit the "I'm here" message from a location token to a context monitor or he can trick some other part of the location system to make the context server believe that the legitimate user is present. If the adversary has the smart card he can now authenticate as the legitimate user.

2. The adversary can perform a *proxy attack* where the adversary tricks the user into using a fake terminal. This allows the adversary to read his password or to perform a man-in-the-middle attack where he can authenticate as the legitimate user without having the user's smart card. This requires that both the user and the adversary is running the same version of the protocol.

   If they are both running the protocol from case A, the adversary simply forwards N to the legitimate user's card, which will return a signature on that value, and the adversary can now send this to the real terminal. The adversary is hereby granted access in the name of the legitimate user. Figure 5.3 illustrates this attack.

   If we are in case B, this is a bit trickier since the adversary must know the correct password in order to succeed. This is because the nonce the real terminal sends is combined with the password and encrypted. In order for this attack to work, the smart card has to sign the nonce the real terminal sent, which means that it must also accept the password entered on the real terminal by the adversary. However this is not a problem for the adversary since the fake terminal can also read the user's password. This attack is also illustrated in Figure 5.3.

5.2.2.3 Summary

It is possible for the adversary to authenticate as a legitimate user by doing one of the following:

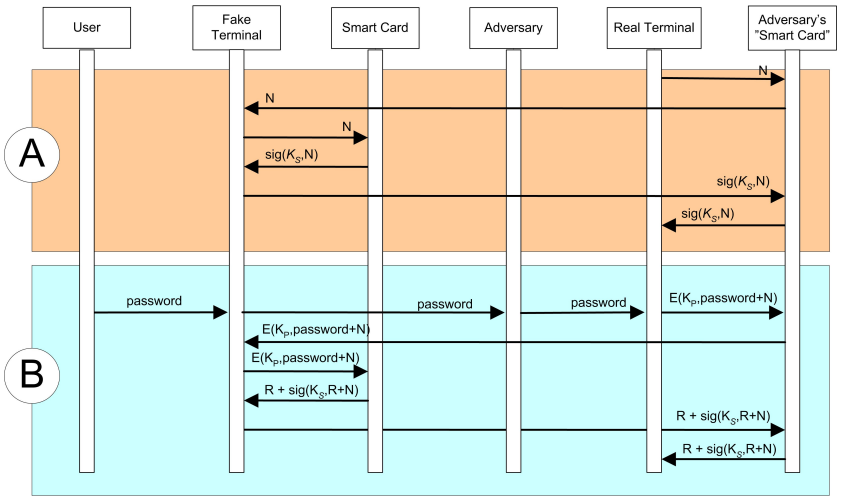


Figure 5.3: Proxy attack where an adversary has put in a fake smart card terminal to be used by an ignorant user.

1. Steal the smart card and fake the location of a legitimate user by
   (a) Replay the ”I’m here” message from a location token
   (b) Trick other parts of the location system in various ways
2. Steal the smart card and be in the same room as the legitimate user.
3. Steal the smart card and acquire the user’s password somehow.
4. Perform one of the two proxy attacks.

We realize that the real weakness in this protocol is the location data. If that can be faked, all you need is to acquire the smart card of a legitimate user, but without knowing the details on how location data is obtained you can not do a thorough security analysis of that problem. If location data is only based on some token the user has, it can be stolen, if such a location token does not use some kind of cryptography it is vulnerable to a reply attack, if you use voice recognition it might be fooled by a recording of the user’s voice, etc. The proxy attacks are bad as well, but require more resources and knowledge to succeed and there are ways to prevent them such as distance bounding protocols [47, 108].

5.2.3 Implementation

Our current implementation consists of five parts:

- An installer.
- The authentication applet.
- A rough prototype of a personal pen

- The context server and monitors
- A client that runs the authentication protocol.

The *installer* installs the applet on the card with the help of IBM JCOP tools [119]. Then it retrieves the public key from the card, and stores it as a key-value pair on the person object in the context server. In our current implementation, the authentication applet uses 512 bit RSA keys. This can easily be changed since the cards also support 1024 and 2048 bit keys. Encryption is done in PKCS#1 mode and signatures are made by taking a SHA1 hash of the data to be signed and then this hash is encrypted with the private key in PKCS#1 mode. We run the applet on the dual-interface OpenPlatform compliant JavaCard designed by IBM, which supports both contact-based and contactless connections to the card reader. We use Philips Semiconductors' MIFARE PRO contactless readers as well as standard OpenPlatform Smart Card readers in e.g. keyboards.

Our current design of the *personal pen* is just to glue a JavaCard to a Mimio Pen, which is used for authentication when using a wall display. This form factor is not particularly appealing, but it is hard to change the form and size of the card, because the antenna is embedded in the plastic that surrounds the chip itself. However, the size of the chip in these cards does not prevent it from being embedded in a pen at the factory. Our ideal hardware design would be a smart card chip embedded in a pen, and the reader embedded in the touch-sensitive layer on a display. Putting the tip of the pen at the display would then correspond to inserting a card in a card reader. In this way we would be able to authenticate the user every time he is doing anything on the display. The *User Authentication Client* runs as a part of the Activity-Based Computing infrastructure, and it simply waits for a card to be inserted in the reader, and then runs the protocol.

As for the *context server*, we have currently implemented the two monitors shown in figure 5.2. The WLAN monitor monitors the WLAN base station infrastructure in our lab and can tell the cell-based location of IEEE 802.11b networked devices. Various types of RFID monitors can monitor passive RFID tags in the environment. We currently use the portal antennas to determine the location of persons equipped with RFID tags.

The current implementation of the *Accuracy Algorithm* is very simple. It reduces the accuracy of the location estimate by 1% every minute. Thus, if a person has passed a portal 10 minutes ago, he is in this location with a probability of 90%. The User Authentication Client is also considered a trusted monitor (we call it a login monitor) and can hence reveal the user's location every time he logs in. The secure authentication of monitors has not been implemented using proper cryptograpic protocols. However, since monitors run on standard PCs there are already well-known ways of doing this using e.g. a PKI setup over secure IP. Hence, this was not necessary in order to make a proof-of-concept.

5.3 The All-Or-Nothing Anti-Theft Policy

In this section we present a security policy we call the All-Or-Nothing security policy, which may help remedy the problem of theft of pervasive computing devices. We describe how this policy can be realized, provide sample applications and documentation that indicate that our solution is realizable on commercial pervasive computing platforms such as cell phones or even smaller devices. A benefit of this policy is that it also provides distribution of cryptographic key material, based on which a security architecture for providing confidentiality or authenticity can be leveraged.

We would like to point out that we focus on key distribution and anti-theft in a network of devices belonging to a single user. Clearly there are many other issues related to this area such as usability, thorough performance evaluation, configuration of devices, management of the network, etc. While we do touch upon some of these issues, they are not thoroughly addressed here.

The starting point of our work, is the vision of pervasive computing described earlier, where people will be surrounded by devices with embedded computers. Pervasive computing devices owned by a particular person will often have (at least) three interesting properties related to our work in this section: 1) they are the target of theft, 2) they have embedded computers, and 3) there are many devices owned by the same person.

5.3.1 Resurrecting Duckling

The overall security policy is modelled after the resurrecting duckling security policy model [179], so we start by briefly reviewing this. The resurrecting duckling security policy model, as described by Stajano and Anderson, consists of four principles:

State Any device is always in one of two states: imprintable or imprinted.

Imprinting An imprintable device is enrolled into a system by receiving some secret. The exchange of this secret is sometimes referred to as a secure transient association [16].

Death The process of going from being imprinted to being imprintable. This is typically, but not necessarily, initiated by the entity (device or human) that imprinted the device.

Assassination Death can only be performed by authorized entities, and assassinating a device, that is an unauthorized change from imprinted to imprintable, should be ”hard”.

The basic idea is that the producer produces devices in the imprintable state. The end-user will purchase the device in this state and when connecting it to his existing equipment, the device will be imprinted. When the user no longer wishes to use a devices (e.g. when it is resold), death will be performed and the device will once again be imprintable. Finally, it must be hard to assassinate imprinted devices, meaning it must be hard to invoke death in an unauthorized

manner. We note that the definition of "hard" in this context is very informal, since the meaning is just that it should cost more resources in terms of time and money, than what the device we are trying to protect is worth.

In the work by Stajano and Anderson [179] imprinting is always done by one device, the mother duck. This imprinting, or pre-authentication [16], is crucial to the realized security of the system, as cryptographic keys are exchanged in this phase. Thus great care should be put into this phase, both from a technical security as well as a usability perspective.

5.3.2 Basic Principle

Our overall idea is to use the embedded computers to cryptographically chain many devices owned by the same person together in a manner which forces a thief to steal *all* of the devices belonging to a particular person, if the thief is to get the desired functionality from these devices.

What this basically means is that a user will enrol all of his devices into a network of friendly devices, and any device on this network will only operate with full functionality if it can "see" all (or sufficiently many) of the other devices in the network. This leads us to the governing principle of our security policy:

- **All-Or-Nothing**: It must be "sufficiently" hard to use stolen equipment in any other configuration than the one it was in when stolen.

Of course, a thief succeeding in stealing all of the devices in a network will have stolen a network of fully functional devices. But, according to the All-Or-Nothing principle the thief cannot (well, it is hard) de-compose the network or remove devices from it. If the thief is to be successful he must steal all devices and keep them together, even when he fences them off to some "customer" who in turn must also keep all the devices together to keep them operational.

This can lead to two different kinds of theft prevention: We either believe that it may be possible to steal all devices in which case focus must be on detecting and recovering from theft, alternatively it might only be (realistically) possible for a thief to steal some, but not all, devices in which case focus must be on detect/recover. We shall go into more details on this subject in the different application domains in Section 5.3.6.

Our second principle may seem rude, but history shows that explicit focus must be kept on this. One example is the X-box that contained a fixed secret key for a symmetric cryptographic algorithm (RC4) [181]. So, our second overall principle is:

- **AntiInduction**: If an attacker manages to break the above property for one device, it must still be equally hard to break another device.

That is, breaking one device does not lead to breaking an entire class of devices. This is related to Kerckhoffs' principle which Shannon reformulated as "The enemy knows the system". We cannot assume that global secrets stay secret forever.

5.3.3 The All-Or-Nothing Security Policy

Realizing the All-Or-Nothing policy means that a device should be unable to function properly when it is removed from its network. Of course this might not always be desirable in a strict sense, since every device would have to be available all the time for this to work. So, we are satisfied with just having some subset of size t of all devices in the system available.

Now we describe our security policy as an extension of the resurrecting duckling security policy [179]. At any given point in time we call the number of devices in the network n.

State Each device is in one of three states:

- **Imprintable** In this state the device is promiscuous and will associate to any network of devices to which it is presented.
- **Imprinted** In this state the device is part of a network of friendly devices $\mathtt{D}_1, ..., \mathtt{D}_n$.

  An imprinted device will occasionally, at least when imprinting another device and when performing an operation like power on, verify the presence of the other members of the network. If it fails to verify this presence it will move to the *emergency* state.
- **Emergency** In this state the device will perform some set of application specific actions to recover. Following these actions, the device will either return to the imprinted or imprintable state. The latter, imprintable, state only following user authentication towards the device, as this amounts to performing death (see below).

Imprinting The user authenticates to some device in the network which then associates with the new device. Afterwards, the new device will automatically associate with the remaining devices in the network.

Death The user authenticates to the device being killed, and the device reverts to the imprintable state.

Assassination It must be "sufficiently" hard to subvert a device from the imprinted to the imprintable state, without authenticating towards the device.

From the above we see that security mechanisms realizing this policy must address the following issues:

- A protocol for device association in the imprinting phase.
- A protocol for presence verification.
- User authentication towards devices.
- Frequency of presence verification.
- Emergency rules.

We describe general application-independent protocols for device association and presence verification, but leave some details on thresholds, user authentication, frequency of presence verification and emergency rules to the specific applications. With respect to user authentication, this means that the user authenticates himself towards the device in some way. Either with a PIN code entered directly on the device, entered on a small portable keyboard that can be attached to the device, read from a biometric scanner, etc.

Note also that a full real-life security architecture must probably also address other attacks than theft. Standard security goals such as confidentiality, etc. may also be of relevance. As mentioned we do not address these issues here, but note that our solution presented is based on equipping all devices with cryptographic keys, which can of course have many uses besides realizing the All-Or-Nothing anti-theft policy. Note that there are various dangers of reusing key material for different purposes, but in our case, the key exchange protocols can just be used to exchange more key material than what is needed for anti-theft, and use it for achieving other security goals.

5.3.4 With Symmetric Keys

Here we describe a design of the All-Or-Nothing anti theft policy, relying only on symmetric key cryptography. The goal is not to provide a complete system ready for deployment, but rather to show that the All-Or-Nothing anti theft policy, can in fact be realized even on resource constrained devices.

5.3.4.1 Protocols

Device Association. The system consists of devices $\mathtt{D}_1, ..., \mathtt{D}_n$ and the new device to be imprinted will be called $\mathtt{D}_m$, where m will be assigned device number $n + 1$.

Imprinting takes place in three phases: First the user transfers a master PIN to $\mathtt{D}_m$ and one other device on the network. We assume without loss of generality that this device is $\mathtt{D}_1$. Second, $\mathtt{D}_1$ and $\mathtt{D}_m$ establish a shared key using a password-based key exchange protocol with PIN as the shared secret. Third, $\mathtt{D}_m$ runs a protocol with the remaining devices to obtain shared keys with them.

We assume that the system is in a state where devices have shared keys with each other, so K_{ij} denotes the key shared by $\mathtt{D}_i$ and $\mathtt{D}_j$ which of course is similar to K_{ji}. Furthermore, device $\mathtt{D}_i$ stores $H(\mathtt{PIN} \| \mathtt{D}_i)$, where $\|$ denotes concatenation and H is a cryptographic hash function.

We make use of two primitives: *authenticated ping* and *presence verification*. Authenticated ping is a symmetric two-way authentication protocol to allow two devices to prove to each other that they know a shared secret key. Presence verification is simply authenticated ping from one device to t other devices, to verify that the device has not been removed from the system. Both are described in more details later in this section. Whenever we say that something is verified, we assume that the device will abort the protocol if verification fails. The first phase is the user phase:

1. The user transfers PIN to $\mathtt{D}_m$.

2. The user transfers PIN to $\mathtt{D}_1$.
3. $\mathtt{D}_1$ verifies PIN using $H(\mathtt{PIN}\|\mathtt{D}_1)$.

The second phase is the key exchange phase. Here $\mathtt{D}_1$ and $\mathtt{D}_m$ exchange a shared key:

1. $\mathtt{D}_1$ sends a random value K_r to $\mathtt{D}_m$.
2. $\mathtt{D}_m$ sends a random value K_s to $\mathtt{D}_1$.
3. $\mathtt{D}_1$ and $\mathtt{D}_m$ compute $K_{1m} = H(\mathtt{D}_1\|\mathtt{D}_m\|\mathtt{PIN}\|K_r\|K_s)$.
4. $\mathtt{D}_1$ runs *authenticated ping* with $\mathtt{D}_m$ to verify K_{1m} and if correct, $\mathtt{D}_1$ accepts $\mathtt{D}_m$ as a new device in the network.
5. $\mathtt{D}_1$ sends $E_{K_{1m}}(H(K_{1i}\|\mathtt{D}_m))$ for all $2 \leq i \leq n$ to $\mathtt{D}_m$.
6. $\mathtt{D}_m$ computes $H(H(\mathtt{PIN}\|\mathtt{D}_i)\|\mathtt{D}_m)$ for all $2 \leq i \leq n$.
7. $\mathtt{D}_1$ and $\mathtt{D}_m$ delete PIN.

The third phase is the key distribution phase, and is executed between $\mathtt{D}_m$ and every other device $\mathtt{D}_2, ..., \mathtt{D}_n$. Here we show the execution between $\mathtt{D}_2$ and $\mathtt{D}_m$. E_K denotes both encryption and authentication under key K, for example by performing encryption in GCM mode [186]:

1. $\mathtt{D}_m$ sends ($\mathtt{D}_1$, K_r) to $\mathtt{D}_2$, where K_r is a random value.
2. $\mathtt{D}_2$ sends a random value K_s is to $\mathtt{D}_m$.
3. $\mathtt{D}_2$ and $\mathtt{D}_m$ compute $K_{2m} = H(\mathtt{D}_2\|\mathtt{D}_m\|H(H(\mathtt{PIN}\|\mathtt{D}_2)\|\mathtt{D}_m)\|H(K_{12}\|\mathtt{D}_m)\|K_r\|K_s)$.
4. $\mathtt{D}_2$ runs *authenticated ping* with $\mathtt{D}_m$ to verify K_{2m} and if correct, $\mathtt{D}_2$ accepts $\mathtt{D}_m$ as a new device in the network.
5. $\mathtt{D}_m$ deletes $H(H(\mathtt{PIN}\|\mathtt{D}_2)\|\mathtt{D}_m)$ and $H(K_{12}\|\mathtt{D}_m)$.

We note that in case some devices are offline during the key distribution phase, $\mathtt{D}_m$ can store the values $H(K_{1i}\|\mathtt{D}_m)$ and $H(H(\mathtt{PIN}\|\mathtt{D}_i)\|\mathtt{D}_m)$ until $\mathtt{D}_i$ becomes available.

Presence Verification. In order to perform an important operation, a device will require permission from t other devices, to make sure that these devices are still around. We call this presence verification with threshold t. If t devices are present, this is a very good indication that the device has not been removed from the system and the operation is allowed.

Call the device that wants to perform the operation $\mathtt{D}_1$ and another device in the system $\mathtt{D}_2$. The following protocol, authenticated ping, now takes place between $\mathtt{D}_1$ and $\mathtt{D}_2$. The protocol is a secure two-way authentication protocol by Bird, Gopal, Herzberg, Janson, Kutten, Molva, and Yung [30], E is a symmetric encryption function in ECB mode using the key shared between $\mathtt{D}_1$ and $\mathtt{D}_2$ and $\oplus$ denotes XOR:

1. $\texttt{D}_1$ says hello to $\texttt{D}_2$.
2. $\texttt{D}_2$ sends a nonce N_2 to $\texttt{D}_1$.
3. $\texttt{D}_1$ replies with $E((\texttt{D}_1 \oplus N_1) \oplus E(N_2 \oplus E(N_2))), N_1$.
4. $\texttt{D}_2$ verifies the value from Step 3.
5. $\texttt{D}_2$ replies with $E(N_2 \oplus E(N_1))$.
6. $\texttt{D}_1$ verifies the value from Step 5.

After completing this protocol with t devices, $\texttt{D}_1$ will assume that it has not been removed from the system, and will continue operating normally.

5.3.4.2 Security Analysis

In this section we argue informally about the security of the scheme with regards to an adversary $\mathcal{A}$ who tries to steal one device from a system of n devices, using a threshold t for authenticated ping.

First we distinguish between corrupted devices and stolen devices. A corrupted device has been tampered with in a way such that the software or hardware running on it can no longer be trusted to perform as expected, and the adversary has access to all data stored on that device. A stolen device is merely removed from the system but will still behave according to the protocol.

We assume that $\mathcal{A}$ is able to eavesdrop on, or alter all communication between devices, and corrupt at most $t - 1$ devices. For simplicity, we assume that the adversary cannot corrupt devices during the user phase and the key exchange phase. It does not seem realistic in a real world setting, as the user is physically in touch with the devices at that time. The goal of the adversary is to steal one of the uncorrupted devices, and make it function as if it was still part of the network. In other words, he must be able to answer at least t authenticated pings correctly.

We start with a list of trivial observations: If $\mathcal{A}$ can corrupt a device, that device can be removed from the system and still maintain full functionality. If $\mathcal{A}$ can corrupt t or more devices, there is a chance that he can answer authenticated ping from a uncorrupted device. Finally, if $\mathcal{A}$ gets access to the `PIN` he has full control over the system.

What the system does ensure is that short obtaining the `PIN` from the user, or using brute force to guess it using data from a corrupted device, the only feasible attacks seem to involve compromising either many devices or the device the adversary wants to steal. The difficulty in compromising a device depends on how many resources are spent securing the device, but even many cheap devices, can help protect a few expensive devices in the system from theft.

Many messages in the protocol contain data combined with the identity of some devices. While this seems to offer little or no protection, since the device identities are known, the idea is to restrict a message to be valid only for a certain set of devices. If this message is retransmitted to another device not in the set, it will be dropped as the message does not match the expected value.

Data Obtained from a Corrupted Device. If $\mathcal{A}$ corrupts a device, he obtains $H(\texttt{PIN}\|\texttt{D}_i)$ where $\texttt{D}_i$ is the identity of the corrupted device. Furthermore he learns the shared keys that this device has with every other device.

This allows him to mount a brute force attack on the PIN by trying all possible values of PIN until one matches $H(\texttt{PIN}\|\texttt{D}_i)$. Choosing a secure PIN and/or a slow hash function, for example by using PKCS#5, reduces the risk of this attack. We note that in this case $\texttt{D}_i$ acts as a salt, making dictionary attacks against the PIN more difficult. Also note that if $\mathcal{A}$ can keep a corrupted device in the system, he gets the PIN if that device is ever used in the user phase.

The keys shared between the corrupted device and all other devices, enables $\mathcal{A}$ to answer one authenticated ping correctly from an uncorrupted device, so as long as he does not corrupt more than t devices, this does not break the system. He could also try to use these keys in the key distribution phase to add his own devices to the system. Why this will fail, we return to later.

Obtaining the PIN. Besides compromising a device and mounting a brute force attack against the PIN, there are two kinds of online attacks against the PIN. The first one is to try to perform the key exchange with a device (Step 1-3 of the key exchange phase), trying different PINs. The opportunity for this attack is severely limited, since he would need to mount the attack while the new device has not yet joined the network. After that, the device forgets PIN and is unable to perform a key exchange. The second online attack is to get physical access to the device and try to enter the PIN pretending to be the user. This is easily mitigated by inserting an increasing delay after each failed attempt.

User Phase. We assume that the devices involved in this phase are not corrupted, otherwise $\mathcal{A}$ would already have access to the PIN. In order to reduce the likelihood of this attack, the user should always use the most secure device to enter his PIN and only use products from respectable vendors. The value $H(\texttt{PIN}\|\texttt{D}_1)$ just enables the device to verify PIN and provide some feedback to the user.

Key Exchange Phase. Security of the newly generated key follows from the key-exchange protocol based on PIN in Step 1-3. Being able to pass authenticated ping in Step 4, proves to $\texttt{D}_1$ that $\texttt{D}_m$ was in possession of PIN, or in other words, that it was authorized by the user to join the network.

In Step 5, $\texttt{D}_1$ sends some data to $\texttt{D}_m$ encrypted under the newly generated key (hence replay attacks do not work, since both devices contribute randomness to the key) and finally $\texttt{D}_1$ does some local computation based on PIN. Nothing here is useful to $\mathcal{A}$.

Key Distribution Phase. One might wonder why the key exchange phase and the key distribution phase are not merged into one phase. The main reason for this is to prevent online attacks against PIN. If we only used $H(\texttt{PIN}\|\texttt{D}_2)$ in

this step, the adversary could try to guess PIN. This is actually an important point. Usually PIN will be rather short (a well known producer of HiFi equipment uses a four digit PIN) and while we can limit the amount of times the user is allowed to enter it on a device, it is more difficult to limit the number of times a device can attempt to perform key exchange. Doing so would make the system highly vulnerable to a denial of service attack.

A passive adversary does not know $H(H(\texttt{PIN}\|\texttt{D}_i)\|\texttt{D}_m)$ or $H(K_{12}\|\texttt{D}_m)$, so he does not get to know K_{2m}. If $\mathcal{A}$ corrupts $\texttt{D}_m$ while it still holds on to some $H(H(\texttt{PIN}\|\texttt{D}_i)\|\texttt{D}_m)$ and $H(K_{1i}\|\texttt{D}_m)$ values (that is, after the key exchange phase, but before the key distribution phase is complete), he gets nothing more than access to one corrupted device, since those values can only be used to make another device accept $\texttt{D}_m$, so once $\texttt{D}_m$ joins the network, those values are of no use to $\mathcal{A}$. However, if $\texttt{D}_m$ is revoked by the user, the adversary can use these values to introduce a new device into the network with the same identity.

To summarize, we use $H(K_{1i}\|\texttt{D}_m)$ in the key distribution phase to prevent online attacks against PIN, and we use $H(H(\texttt{PIN}\|\texttt{D}_i)\|\texttt{D}_m)$ to prevent $\mathcal{A}$ from using key material from one corrupted device to add other devices to the system. It doesn't matter if $\mathcal{A}$ corrupts one of the devices between the key exchange phase and the key distribution phase, or after the device finishes joining the network. The end result is that he has only one more corrupted device in the network.

Authenticated Ping. Security of authenticated ping, and therefore also presence verification, follows from the security of the two-way authentication protocol by Bird et al. [30].

Proxy Attack. A specific type of attack against such authentication systems is the proxy attack, also known as the mafia fraud attack. In this attack an adversary would steal a device and install a proxy device in the network. This proxy device would relay communication between the network and the stolen device over a long distance, meaning that the stolen device could always perform a valid authenticated ping. Such attacks are usually always possible, but may be countered by distance bounding protocols [47] which may even be realized on resource constrained platforms [108]. While the proxy attack is devastating in some scenarios, for theft protection it does not seem feasible. First of all the adversary has to leave a device in the network he stole a device from and second, he needs to be able to answer authenticated pings all the time. If the proxy device at some point is discovered and removed, his stolen device stops working. We will discuss the implications of this attack in the scenarios in Section 5.3.6.

Summary. We conclude that short of obtaining the PIN the only feasible attack seems to involve compromising either many devices or the device the adversary wants to steal, which can be made difficult by using secure hardware. While not all devices carry a price tag that encourages the use of secure hardware, even many cheap devices, can help protect a few expensive devices in

the system. For example, the coffee machine, and a couple of light bulbs and on/off switches, can protect the expensive home entertainment system, and if the devices are already in a network where cryptographic keys are needed, then there is very little overhead in also using these keys for theft protection.

5.3.4.3 Implementation

The purpose of our implementation efforts was to see whether or not this idea is realizable in practise, not to implement the full scheme. We chose to implement authenticated ping since it is a building block used a lot, and performance-wise none of the other protocols seem much harder to execute, so we believe it is a good indication of the performance of the complete scheme.

We have chosen to use motes for our implementation as they represent one of the more resource constrained platforms which might form the basis for some of our proposed applications. So, we argue that if our solution is viable on this platforms, it also is on more powerful ones. We use TMote Sky motes from Moteiv which are commercially available.

The hardware was four TMote Sky, with a 16-bit 8MHz TI MSP430 CPU, 10kb of RAM, 48kb of flash memory, and a 250kbps 2.4GHz IEEE 802.15.4 wireless transceiver. They have three LEDs and two buttons, where one is the reset button. Development was done on TinyOS 1.x which takes care of mesh networking and communication. The programming language was NesC.

We implemented authenticated ping on the motes with a threshold of $t = 2$, meaning that if one of the motes was down, or communication failed for some reason, the ping would still succeed. Since we did not implement any key exchange, the encryption keys were hard-coded at development time. For symmetric encryption we implemented the XTEA algorithm with 32 rounds. We experimented with smaller number of rounds, but it seemed to have a negligible impact on performance. However, to save power on battery-operated devices, the number of rounds can be reduced since the data encrypted during authenticated ping, only needs to remain secure for a short period of time. XTEA uses 128-bit keys and 64-bit blocks, which had to be taken into account, since the maximum number of bytes TinyOS supports in a message is 29.

When the user button was pressed, the mote would perform presence verification and the green LED would light up if it succeeded, otherwise the red LED would light up. Since the motes can't generate random data we used a PRNG seeded with the mote's address, which does not provide cryptographic random data, but was fine enough for our proof of concept. Another problem was related to interrupt handling on the motes which caused messages to be dropped if two messages arrived while the mote was performing some computation. We circumvented this problem by inserting a small fixed delay between each attempted authenticated ping, giving the previous one time to finish before starting the next.

The binary uploaded to the mote was around 10kb, including everything (TinyOS, our application, libraries, etc.) and used around 500 bytes of RAM when running. Since the priority was just to get it running, there is room for improvement. Especially the code size can be made smaller, but also the

RAM usage. A rough estimate is that the authenticated ping took around 10ms to perform, so for all practical purposes presence verification succeeded immediately when the button was pressed.

5.3.5 With Public Keys

The protocol described in the previous section suffers from one significant drawback, namely that if the `PIN` is revealed, every device is corrupted. This would not be so bad if the `PIN` was well protected, but entering the `PIN` on just one corrupted device, would compromise the entire system. Furthermore, a hash of the `PIN` is stored on every device, which makes it vulnerable to a brute force attack. Due to usability issues, in real life implementations the `PIN` might be rather short, and this might make a brute force attacks feasible.

In this section we present alternative protocols, realizing the All-Or-Nothing security policy. However, this implementation requires slightly more computational power available on the devices, as well as a more advanced device which will act as the ”key” to the system.

5.3.5.1 Prerequisites

We let the user be in possession of a special device, called the `UserToken`, which can communicate with a device over either a physical connection, or a short-ranged wireless technology such as RFID or IR. This token can be considered a ”key” to the system, as the user will require this device to add or remove devices from his system. The `UserToken` stores the following information:

- A private/public keypair (sk, pk) for a digital signature scheme.
- A bitvector of length no less than n, where n is the number of devices in the system, which is known by the `UserToken` since it has to authorize each new device. If $b_i = 1$ then device $\mathtt{D}_i$ is in the network, otherwise it is not. This vector is called the `DeviceList` and has a version number called the `DeviceListVersion`, which is increased by one every time the `DeviceList` changes. Note that there will only be a 0 in the list, when a device is removed from the network.
- A `TimeStamp` that is initialized to 0 and increased by one for every use. Whenever a device receives a `TimeStamp` from the `UserToken`, it will store this value, and not accept any other message from the `UserToken` unless the `TimeStamp` is larger than the last known value.

From now on we will write `UT` for the `UserToken`, `DL` for the `DeviceList`, `DLV` for the `DeviceListVersion`, and `TS` for the `TimeStamp`.

5.3.5.2 Protocols

Again we make use of the same two primitives, *authenticated ping* and *presence verification*, as described in the previous section. Imprinting also takes place in the same three phases as before: First the `UT` generates a new random device

key, which is transferred to $\mathtt{D}_m$ and one other device on the network. We assume without loss of generality that this device is $\mathtt{D}_1$. Then $\mathtt{D}_1$ and $\mathtt{D}_m$ exchange data that allows $\mathtt{D}_m$ to exchange key material with all other devices in the system. Third, $\mathtt{D}_m$ runs a protocol with the remaining devices to obtain shared keys with them.

Recall that the system consists of n devices $\mathtt{D}_1, ..., \mathtt{D}_n$ and the new device to be imprinted will be called $\mathtt{D}_m$ where $m = n + 1$.

Device Association. Let $\mathtt{D}_i$ denote device number i. We assume that the system is in a state where devices have shared keys with every other device, so K_{ij} denotes the key shared by $\mathtt{D}_i$ and $\mathtt{D}_j$ which of course is similar to K_{ji}. Furthermore, device $\mathtt{D}_i$ stores pk, DL, DLV and TS.

Again, when we say that something is verified, we assume that the device will abort the protocol, and forget all data received during the protocol if verification fails. We also assume that a device will refuse to negotiate a new key, if it already shares a key with that device. The first phase is the user phase:

1. UT generates a random key K_t.
2. UT sends (pk, K_t) to $\mathtt{D}_m$.
3. UT updates the DL to contain $\mathtt{D}_m$.
4. UT sends $\mathtt{DLV}, \mathtt{DL}, Sig_{sk}(\mathtt{DLV}, \mathtt{DL})$ to $\mathtt{D}_m$.
5. UT sends $\mathtt{TS}, K_t, \mathtt{D}_m, Sig_{sk}(\mathtt{D}_1, \mathtt{TS}, K_t, \mathtt{D}_m)$ to $\mathtt{D}_1$.
6. $\mathtt{D}_1$ verifies the signature.

The second phase is the key exchange phase, where $\mathtt{D}_1$ and $\mathtt{D}_m$ exchange a new shared key. The DL is updated by the authenticated ping protocol, as we will see later.

1. $\mathtt{D}_1$ sends a random value K_r to $\mathtt{D}_m$.
2. $\mathtt{D}_m$ sends a random value K_s to $\mathtt{D}_1$.
3. $\mathtt{D}_1$ and $\mathtt{D}_m$ compute $K_{1m} = H(\mathtt{D}_1 \| \mathtt{D}_m \| K_t \| K_r \| K_s)$.
4. $\mathtt{D}_1$ runs *authenticated ping* with $\mathtt{D}_m$ to verify K_{1m}.
5. $\mathtt{D}_1$ checks that $\mathtt{D}_m$ is in the DL.
6. $\mathtt{D}_1$ sends $E_{K_{1m}}(H(K_{1i} \| \mathtt{D}_m))$ for all $2 \leq i \leq n$ to $\mathtt{D}_m$.

The third phase is the key distribution phase, and is executed between $\mathtt{D}_m$ and every other device $\mathtt{D}_2, ..., \mathtt{D}_n$. Here we show the execution between $\mathtt{D}_m$ and $\mathtt{D}_2$

1. $\mathtt{D}_m$ sends ($\mathtt{D}_1$, K_r) to $\mathtt{D}_2$, where K_r is a random value.
2. $\mathtt{D}_2$ sends a random value K_s is to $\mathtt{D}_m$.

3. Both compute $K_{2m} = H(\mathtt{D}_2 \| \mathtt{D}_m \| H(K_{12} \| \mathtt{D}_m) \| K_r \| K_s)$.
4. $\mathtt{D}_m$ runs *authenticated ping* with $\mathtt{D}_2$ to verify K_{2m}.
5. $\mathtt{D}_2$ checks that $\mathtt{D}_m$ is in the $\mathtt{DL}$.
6. $\mathtt{D}_m$ forgets $H(K_{12} \| \mathtt{D}_m)$.

We note that in case some devices are offline during the key distribution phase, $\mathtt{D}_m$ can store the value $H(K_{1i} \| \mathtt{D}_m)$ until $\mathtt{D}_i$ becomes available. Again E_K denotes both encryption and authentication under key K.

Presence Verification. Call the device that wants to perform the operation $\mathtt{D}_1$ and another device in the system $\mathtt{D}_2$. The following protocol, authenticated ping, now takes place between $\mathtt{D}_1$ and $\mathtt{D}_2$:

1. $\mathtt{D}_1$ sends $\mathtt{DLV}_1$ to $\mathtt{D}_2$

`If` $\mathtt{DLV}_1 > \mathtt{DLV}_2$:

2. $\mathtt{D}_2$ requests the updated $\mathtt{DL}$ from $\mathtt{D}_1$
3. $\mathtt{D}_1$ sends $\mathtt{DLV}_1, \mathtt{DL}_1, Sig_{sk}(\mathtt{DLV}_1, \mathtt{DL}_1)$ to $\mathtt{D}_2$
4. $\mathtt{D}_2$ verifies the signature
5. $\mathtt{D}_1$ restarts *authenticated ping* with $\mathtt{D}_2$

`else if` $\mathtt{DLV}_1 < \mathtt{DLV}_2$:

2. $\mathtt{D}_2$ sends $\mathtt{DLV}_2, \mathtt{DL}_2, Sig_{sk}(\mathtt{DLV}_2, \mathtt{DL}_2)$ to $\mathtt{D}_1$
3. $\mathtt{D}_1$ verifies the signature
4. $\mathtt{D}_1$ restarts *authenticated ping* with $\mathtt{D}_2$

`else`:

2. $\mathtt{D}_2$ sends a nonce N_1 to $\mathtt{D}_1$
3. $\mathtt{D}_1$ replies with $E(\{\mathtt{D}_1 \ \oplus N_1\} \oplus E(N_0 \oplus E(N_1))), N_0$
4. $\mathtt{D}_2$ verifies the value from the previous step
5. $\mathtt{D}_2$ replies with $E(N_0 \oplus E(N_1))$
6. $\mathtt{D}_1$ verifies the value from the previous step

The difference from the version in the previous section is that we need to update the $\mathtt{DL}$ which will be interleaved with presence verification, otherwise it is the same two-way authentication protocol.

Updating the DL. When a new device is added to, or removed from, the system, we need to make all other devices aware of this fact. This is done by updating the DL on each device. Since the DL is generated by the UT and carries a version number, it is easy to determine which version is the latest. The DL is updated on the devices by the authenticated ping protocol when a new device is added, but we also must be able to remove devices from the system. Assume we want to remove $\mathtt{D}_1$ from the system, and that $\mathtt{D}_2$ is a device still in the system.

1. UT sends $\mathtt{TS}, Sig_{sk}(\mathtt{TS}, \mathtt{D}_1)$ to $\mathtt{D}_1$
2. $\mathtt{D}_1$ verifies the signature and becomes imprintable
3. UT removes $\mathtt{D}_1$ from the DL.
4. UT sends $\mathtt{DLV}, \mathtt{DL}, Sig_{sk}(\mathtt{DLV}, \mathtt{DL})$ to $\mathtt{D}_2$
5. $\mathtt{D}_2$ verifies the signature
6. $\mathtt{D}_2$ performs authenticated ping with $t > n/2$

5.3.5.3 Security Analysis

The protocols here are in fact similar to those in Section 5.3.4.2. The main difference is that there is no way to perform a brute force attack against a user defined value such as the PIN. Lacking the PIN, we don't have the same way of ensuring that keys from one corrupted device are not used to add a number of other devices to the network. However, in this case we have the DL which is generated by the UT, so if a device does not appear on the DL it is not allowed to join the network. This ensures that only devices authorized by the user can join (or one other device, namely $\mathtt{D}_m$, if $\mathcal{A}$ corrupts $\mathtt{D}_m$ between the key exchange phase and the key distribution phase, but in this case the user can just remove $\mathtt{D}_m$ from the network when he realizes things are not working as expected). Furthermore it has the benefit that it is easier to keep track of devices in the network.

This design also thwarts the attack, where $\mathcal{A}$ corrupts e.g. $\mathtt{D}_1$, and uses key material from it to enroll $\mathtt{D}_m$ in the network after the real $\mathtt{D}_m$ has been removed by the user for some reason. This attack worked against the protocol from Section 5.3.4 but doesn't work here, because the UT has to accept a device by updating the DL, where the previous version depended on devices themselves to announce a new device to the network.

To summarize, this protocol provides a number of benefits over the previous version from Section 5.3.4. It does not rely on a global secret that can be guessed based on data stored on each device, and it allows a more consistent view of which devices belong to the network. In the previous version, devices were added to the network whenever they contacted a device to exchange keys. In this protocol, there is a global list of devices in the network.

5.3.5.4 Performance and Usability

UserToken. We envision the UT as a small device with either a very short range radio (a few centimeters) or as a pen-like device where the tip of the pen can touch a small area on a device, thereby creating physical contact between them for transferring data. The UT could be equipped with only one LED for informing the user, and two buttons. One for adding a device, and one for revoking a device. Adding a device to the system, could be done in the following way:

1. The user presses the *add device* button and the UT generates a random key K_r. The LED lights up, and this completes Step 1 of the user phase.
2. The user touches the new device, the devices communicate and the LED starts flashing on the UT. This completes Step 2-4 of the user phase.
3. The user touches an existing device in the system, the devices communicate, and the LED stops flashing. This completes the final steps of the user phase.

All the remaining actions can now be completed by the system itself, without the UT. Removing a device would work in the exact same way, except the user would press the *remove device* button.

There might need to be some way of authenticating the user towards the UT before it can be used[1]. This could be done by equipping the UT with a fingerprint reader, or another forms of biometric sensor that could recognize the user.

Backup. Of course the UT will eventually be lost, and a system should be designed to be able to recover from such an event. Notice that the only thing that would need to be backed up from the user token is the public/private keypair, as the other data can be obtained from the devices in the system. If the UT is equipped with some biometric reader, a physical trait of the user could be used to initialize a PRNG and then the same keypair could be regenerated. However one should be aware of the risks associated with depending on biometric traits to generate key material. In other cases, a regular backup procedure (e.g. to the user's PC protected by a password) could be used.

Public Key Algorithm. Many devices will not have the computing power available to fully support public key algorithms, so the protocols have been designed in such a way that the UT generates signatures and the devices only verify them. This allows us to use a signature scheme where signing is expensive, but verification is cheap. One promising candidate is the Rabin signature scheme [164]:

Gen Choose two random primes p, q and compute their product $n = pq$. The secret key is (p, q) and the public key is n.

[1] In many scenarios, it probably wouldn't be necessary. After all, people are quite used to handle regular keys, which can be used by anyone who physically gets hold of them.

Sign To sign a message m, choose a random u such that $|u| \approx 60$ bits, and compute $c = H(m\|u)$. Check that $gcd(c, n) = 1$ and that c is a quadratic residue modulo n. If this test fails, pick a new random u and try again. Now compute $x = \sqrt{c} \bmod n$. Output the signature (x, u).

Verify To verify a signature (x, u) on a message m, check if $H(m\|u) = x^2 \bmod n$. Output *accept* if this holds, otherwise output *reject*.

Verifying the signature requires just one modular multiplication and one evaluation of a hash function, which certainly is feasible on, for example, off the shelf ZigBee devices.

5.3.6 Applications

In this section we study three different applications of the All-Or-Nothing anti-theft policy. Clearly we can use both the symmetric and the public key variant of the protocol, since the main difference to the user, is how imprinting and management of the network is done. A general guideline could be that in networks with many devices at home, the `UT` approach makes sense, whereas it does not make much sense if a single manufacturer decides to use his own system, that is incompatible with most other devices the user has in his possession.

In an ideal world, the user would have just one `UT` that is used to imprint all his devices at home, but if the All-Or-Nothing policy is applied to other scenarios (as we will see in this section), we will have systems that are independent of each other. In that case, having one `UT` for each system might not be the best idea from a usability point of view, and also it might be too costly for the manufacturer to provide such a hardware device to the user.

Since use cases for devices in the user's home, all imprinted by the same `UT` is quite straightforward, we will focus our attention on applying the policy in other settings. Hence we do not consider the `UT` approach in the following.

Entertainment Systems. Theft is a huge problem, especially for high value products such as modern entertainment systems. There exist sophisticated burglar rings gathering intelligence on what equipment can be stolen where and use this knowledge to obtain, on demand, whatever items their "customers" have on their shopping list. Such intelligence could come from insiders at stores that sell such equipment, burglars can drive around residential areas at night with product specific remote controls to find out where certain (types of) devices can be found, etc.

Of course it is impossible to completely prevent theft, hence the strategy often employed is to minimize the value of stolen equipment by maximizing the efforts needed to get stolen equipment to function as if it was acquired legitimately. In other words, our security goal is to make using stolen devices as troublesome as possible, which in the context of the All-Or-Nothing policy translates to forcing a thieve to steal and fence entire systems rather than single devices, or to spend more resources trying to corrupt a device, than what the device is worth.

Some special features of devices in such systems are that they are not mobile, and they may not all have specific input devices (e.g. loudspeakers). We shall assume that the devices have plenty of computing power (comparable to, say, a cheap PC), are networked (i.e., communicating via a digital bus), and that all devices are equipped with a USB-port. In some cases it may actually be desirable to enforce that devices originate from the same manufacturer. To achieve this all devices could be equipped with a root certificate from the producer together with a certified public-key pair, that will be used during the imprinting protocol. We note that this would be entirely optional.

As we have a relatively static set of non-mobile devices, we shall set the threshold to be the total number of devices minus one, $t = n - 1$, such that any device must be able to see all other devices[2]. Note that this threshold only works if we assume a closed system consisting only of the home entertainment system. If devices from other domains are also added to the system, we need to define another threshold.

We imagine a scenario where the maker of the home entertainment system sets up their own anti-theft solution, independent of other manufacturers. As mentioned in the beginning of this section, that makes the `UT` approach less desirable, but nevertheless it allows some manufacturers to implement such an anti-theft solution today. Therefore we assume that user authentication will be based on the user having a special USB PIN-pad. This will be connected to the relevant devices when PINs are to be entered. We imagine that the user receives this low cost unit when he purchases his first device.

The frequency of presence verification is defined so that in addition to when doing imprinting, we perform presence verification whenever a device has been without electricity. The reason is that a thief must unplug a device to steal it.

Finally, we define the following emergency rules. 1) If presence verification fails the device will request user authentication to perform death. If this fails, the device will shutdown for a period of time (this period of time should increase for each failure), after which the device will again request user authentication to perform death. 2) If the remaining devices reappear, the device will also require user authentication with the same increasing shutdown period, before the device can return to the imprinted state and normal operation can be resumed. In this manner, all devices must be stolen simultaneously without any interruption in the power supply if the thief is to have any functionality.

In this scenario, the proxy attack does not seem very likely as the owner of the devices will know immediately that devices have been stolen. If some devices have been stolen, the remaining devices should not work anymore. If they do it is a sure sign that something is wrong. We note that even if one device is stolen, the user can still perform death on the remaining devices and add them to a new system.

Personal Devices. Recently Sony Ericsson released a Bluetooth watch which can control cell phones and show information such as caller ID. Further, you

[2]To allow the owner to take single devices to service etc., one could perhaps work with a threshold of $t = n - 2$ or $t = n - 3$.

may configure the watch to issue an alarm if it looses connection to the cell phone. There is good reason for such a feature as cell phones, PDAs and other personal pervasive computing devices are often stolen or forgotten. We now apply the All-Or-Nothing policy to such personal devices.

Devices such as cell phones or Bluetooth enabled watches cannot be assumed to have plenty of computing power, but as our implementation experiments show, they have sufficient power for our needs. In contrast to entertainment systems they are highly mobile, albeit always close to a particular person and thus close to each other. Also, we will assume that devices either have input devices themselves or can be configured over Bluetooth from another device.

In this case we focus on a user with just a few devices (such as a watch, a cell phone, and a PDA, $n = 3$) and set the threshold to $t = 1$. This means that for a device to successfully perform presence verification, at least one other device must be present. Since we are dealing with devices that have access to a keyboard (locally or via e.g. Bluetooth) we will use the PIN version of the protocol, since introducing an additional device to imprint just a few personal devices would be overkill.

User authentication is done by directly entering a PIN on each device. This is standard for devices such as cell phones and PDAs. For watches and similar, user authentication could be done by having the user enter the PIN through, say, the cell phone, over Bluetooth, which can then be displayed on the watch and acknowledged through the push of a button. At first this approach may seem insecure, but done in the privacy of your home there is little practical risk involved: If we can make sure that the right PIN is entered, it seems unlikely that a pickpocket will be monitoring Bluetooth traffic in your home.

In this case we choose a high frequency for presence verification, as you want to know immediately if you have forgotten your PDA in a taxi, or if someone has taken your cell phone from your pocket.

We define the following emergency rules: 1) If a device in the emergency state is able to perform presence verification it returns to the imprinted state and resumes normal operation. 2) Otherwise death must be performed by authenticating as the user by entering the PIN. We have two alternative rules if the user fails an authentication. Either, the device will shutdown for increased periods of time as suggested for entertainment systems, or after a fixed number of failures the devices will erase all data and become imprintable or locked. The latter rule should preferably be combined with a sound back-up policy. It might also be made more strict for devices containing sensitive information, e.g. classified business data, in which case data could be erased immediately when entering the emergency state.

One benefit of employing the All-Or-Nothing policy in this scenario is that a user might not need to protect his cell phone or PDA with a PIN, i.e. having to enter the PIN each time he desires to use the device. Since users often do not use the PIN features in cell phone anyway (see e.g. Dourish, Grinter, Flor, and Joseph [85]), the All-Or-Nothing might still provide better practical security than the PIN solution.

In addition to watches, cell phones and PDAs one could of course envision many other devices in the personal sphere which could be tied to a particular

individual. Jewelry with small embedded computers, etc.

Cars. Our final example is theft protection for cars. As described in [182] there may be multiple back doors and attack opportunities for the social engineer. Further, realizing the assassination principle, i.e., founding security in a tamper resistant computing base is notoriously hard [4]. The All-Or-Nothing policy does not specify how to deal with these issues, so we assume that key material etc. is handled diligently and is sufficiently well protected.

Car theft protection is often referred to as immobilizers as they make the car immobile. A widespread current technology is the Digital Signature Transponder (DST) proved cryptographically insecure due to weak proprietary algorithms and too short a key length by Bono, Green, Stubblefield, Juels, Rubin, and Szydlo [38]. Another solution called KeeLoq which was used by Toyota, Honda, Chrystler and Jaguar, just to mention a few, was proven insecure by Biham, Dunkelman, Indesteege, Keller, and Preneel [29]. The solution we propose will behave very similarly to the DST solution, but with our algorithms available for public scrutiny.

An obvious solution would be to simply enrol the car in the network of personal devices described above. If there is more than one user of the car, we can extend the policy to allow for multiple users of each device.

Alternatively, and perhaps more suited to the use of most drivers, we could root the policy in the car and the car keys. The threshold would be $t = 1$, i.e., both the car and the car key (assuming that there is only one key) must be present. The user authentication would be done by the manufacturer, perhaps with a possibility for the car owner choose a new PIN (and new keys) using a special purpose devices as the one used for entertainment systems. It is worth pointing out, that we are not interested in enrolling the car and the car keys into a network of devices at home, since they will not be kept together.

Other that empowering the user to provide his devices with new keys, this might come in handy if the car key is stolen and you want to make sure that the thief cannot later steal the car as well, in which case you perform death on the car and imprint with a new (car) key.

With respect to frequency and emergency rules, this instantiation of the policy is a bit special. The reason is that the car key is more or less passive. Fortunately it is not the key we wish to protect. The car will verify presence whenever someone is attempting to turn the car on. The emergency rules are simple, the car requires user authentication to perform death, but will automatically revert to normal operation whenever presence is reestablished. In this way the car will actually be in the emergency state much of the time (whenever the key is not in the car).

The more politically correct would probably also argue for introducing an alcoholometer in the car which will only confirm its presence if the driver authenticates by breathing into the devices (and is not intoxicated).

Chapter 6

Batch Verification of Signatures

In this chapter we consider the suitability of public key signatures to the needs of some pervasive communication applications.

We start by motivating why efficient verification of many signatures is important, and describe related work, in Section 6.1. In Section 6.2 we introduce some definitions and propose methods for batch verifying an existing signature schemes, and also propose a new signature scheme which on paper is efficient to batch verify. Then we take a more practical approach in Section 6.3. We generalize many of the results from Section 6.2 and implement several schemes to verify the efficiency claims.

The results in this chapter are based on a paper presented at the Eurocrypt 2007 conference [54] as well as the full version [55], and a paper which as the time of writing has not yet been published. [91].

6.1 Introduction

As the world moves towards pervasive computing and communication, devices from vehicles to dog collars will soon be expected to communicate with their environments. For example, many governments and industry consortia are currently planning for the future of *intelligent cars* that constantly communicate with each other and the transportation infrastructure to prevent accidents and to help alleviate traffic congestion [64, 175]. Raya and Hubaux suggest that vehicles will transmit safety messages every 300ms to all other vehicles within a minimum range of 110 meters [165], which in turn may retransmit these messages.

For such pervasive systems to work properly, there are many competing constraints [64, 120, 165, 175]. First, there are physical limitations, such as a limited spectrum allocation for specific types of communications and the potential roaming nature of devices, that require that messages be kept very short and (security) overhead be minimal [120]. Yet for messages to be trusted by their recipients, they need to be authenticated in some fashion, so that entities spreading false information can be held accountable. Thus, some short form of authentication must be added. Third, different messages from many different signers may need to be verified and processed quickly (e.g., every 300ms [165]).

A fourth constraint in some scenarios, is that these authentications remain anonymous. However, not every scenario demands perfect anonymity. In some cases users should be held accountable, but just not become publicly identifiable.

Generating one signature every 300ms is not a problem for current systems, but transmitting and/or verifying 100+ messages per second might pose a problem. Using RSA signatures seems attractive as they are verified quickly, however, one would need approximately 3000 bits to represent a signature on a message plus the certificate (i.e., the public key and signature on that public key) which might be too much for some applications (see Section 8.2 of [165]). While many new schemes based on pairings can provide the same security with significantly smaller signatures, they also take significantly more time to verify. Thus, it is not immediately clear what the proper tradeoff between message length and verification time is for many pervasive communication applications. However, in some applications, there is evidence that doing a *small* amount of additional computation is more advantageous than sending longer messages. For example, Landsiedel, Wehrle, and Götz showed that for applications using Mica2 sensors transmitting data consumes significantly more battery power than keeping the CPU active [133], and Barr and Asanović note that wireless transmission of just a single bit, can use more than 1000 times the energy required for a 32 bit computation [21].

Due to the high overhead of using digital signatures, researchers have developed a number of alternative protocols designed to amortize signatures over many packets [98, 142], or to replace them with symmetric MACs [161] such as HMAC [131]. Each approach has significant drawbacks; for example, the MAC-based protocols use time-delayed delivery so that the necessary verification keys are delivered *after* the messages arrive. This approach can be highly efficient within a restricted setting where synchronized clocks are available, but it does not provide other desirable features such as non-repudiation of messages (to hold malicious users accountable) or privacy. Signature amortization requires verifiers to obtain many packets before verifying, and is vulnerable to denial of service. It is interesting to note that the short, undeniable signatures of Monnerat and Vaudenay [147, 148] support a form of batch verification. However, these are inappropriate for the pervasive settings we consider, since verification is not universal and requires interaction with the signer.

Fast verification of many signatures is an interesting problem in other scenarios as well. Consider a scenario where a mail server receives a lot of signed e-mails. To handle a variety of different e-mail clients on the internal network, it is easier to let the server do signature verification and insert a message into the body of the e-mail about who signed it. Assuming the internal network and the mail server are secure, clients can rely on the signature being correct without having to verify it themselves. However, the actual digital signature can still be attached to the e-mail should a dispute about the authenticity of the message later arise. To keep resource usage on the server to a minimum, signature verification should be fast, but we can take advantage of the fact that the server can buffer messages for a short period before verifying all of them.

If one wants both short signatures and short verification times, it seems that one needs to improve on the verification time of pairing based schemes, or try to reduce the signature size of a fast signature scheme. In this chapter we take the first approach, and investigate the known batch-verification techniques and to what extent they are applicable to pairing based schemes, whereas for example Gentry takes the other approach and provides a method for compressing Rabin signatures [99]. We note that while these two techniques are not mutually exclusive (in fact Gentry mentions that the compressed Rabin signatures can be aggregated [99]), compressing signatures has not been the focus of our work. We begin with an overview of batch verification history.

6.1.1 History

Batch cryptography was introduced in 1989 by Fiat [92] for a variant of RSA. Later, in 1994, Naccache, M'Raïhi, Vaudenay, and Raphaeli [150] gave the first efficient batch verifier for DSA signatures, however an interactive batch verifier presented in an early version of their paper was broken by Lim and Lee [137]. In 1995 Laih and Yen proposed a new method for batch verification of DSA and RSA signatures [132], but the RSA batch verifier was broken five years later by Boyd and Pavlovski [40]. In 1998 Harn presented two batch verification techniques for DSA and RSA [109, 110] but both were later broken [40, 117, 118]. The same year, Bellare, Garay, and Rabin took the first systematic look at batch verification [23] and presented three generic methods for batching modular exponentiations, called the *random subset test*, the *small exponents test* and the *bucket test* which are similar to the ideas from [132, 150]. They showed how to apply these methods to batch verification of DSA signatures and also introduced a weaker form of batch verification called *screening*. In 2000 some attacks against different batch verification schemes, mostly ones based on the small exponents test and related tests, were published [40]. These attacks do not invalidate the proof of security for the small exponents test, but rather show how the small exponents test is often used in a wrong way. However, they also describe methods to repair some broken schemes based on this test. In 2001 Hoshino, Masayuki, and Kobayashi [116] pointed out that the problem discovered in [40] might not be critical for batch verification of signatures, but when using batch verification to verify for example zero-knowledge proofs, it would be. In 2004 Yoon, Cheon, and Kim proposed a new ID-based signature scheme with batch verification [73], but their security proof is for aggregate signatures and does not meet the definition of batch verification from [23]; hence their title is somewhat misleading. Other schemes for batch verification based on pairings were proposed [65, 192–194] but all were later broken by Cao, Lin and Xue [63]. In 2006, a method was proposed for identifying invalid signatures in RSA-type batch signatures [136], but Stanek [180] showed that this method is flawed.

6.1.2 Techniques by Bellare, Garay and Rabin

Let g generate a group of prime order. In 1998, Bellare et al. described some tests [23], for verifying equations of the form $y_i = g^{x_i}$ for $i = 1$ to n, which we will use again in our work. Obviously if one just multiplies these equations together and checks if $\prod_{i=1}^{n} y_i = g^{\sum_{i=1}^{n} x_i}$, it is easy to produce two pairs (x_1, y_1) and (x_2, y_2) such that the product of them verifies correctly, but each individual verification does not, e.g. by submitting the pairs $(x_1 - \alpha, y_1)$ and $(x_2 + \alpha, y_2)$, for any α, instead. Let us review three fixes to this broken proposal.

Random Subset Test The first idea is to pick a random subset of these pairs (x_i, y_i) and multiply them together, hoping to split up the pairs that were specifically crafted to cancel each other out. Repeating this test ℓ times, picking a new random subset every time, results in the probability of accepting invalid pairs being $2^{-\ell}$.

Small Exponents Test Instead of picking a random subset every time, one can instead choose exponents δ_i of (a small number of) ℓ bits and compute $\prod_{i=1}^{n} y_i^{\delta_i} = g^{\sum_{i=1}^{n} x_i \delta_i}$. They also prove that this test results in the probability of accepting a bad pair being $2^{-\ell}$. The size of ℓ is a tradeoff between efficiency and security and hence it is difficult to give an exact recommendation for it. It all depends on the application and how critical it is not to accept even a single invalid signature. For just a rough check that all signatures are correct 20 bits seems reasonable. In a higher security setting we should probably be using around $64 - 80$ bits.

Bucket Test Finally, a method called the *bucket test* is even more efficient than the small exponents test for large values of n. The idea is to repeat a test called the *atomic bucket test* m times. The atomic bucket test works by first putting the n instances one wants to verify into M buckets at random. This results in M new instances of the same problem, which are then checked using the small exponents test with security parameter m. After repeating the atomic bucket test m times, the probability of accepting a bad pair in the original n instances is at most 2^{-m}.

6.2 Batch Verification of Short Signatures

In this section we instantiate the general batch verification definitions of Bellare, et al. [23] to the case of signatures from many signers, where the definitions in [23] only considered a single signer. We also do this for a weaker notion of batch verification called *screening* and show the relation of these notions to the one of aggregate signatures. Surprisingly, for most known aggregate signature schemes a batching algorithm is provably *not* obtained by aggregating many signatures and then verifying the aggregate.

We present a batch verifier for a variant of the Waters IBS scheme [67]. (More precisely, this IBS scheme is implicitly defined by the Chatterjee-Sarkar hierarchical IBE [67] and it can also be viewed as a generalized version of the Boyen-Waters IBS [42] as we will discuss later.) To our knowledge, this is

the first batch verifier for a signature scheme without random oracles. Let z be the additional security parameter required by the generalization. When identities and messages are k bits, viewed as z chunks of k/z bits each, our algorithm verifies n signatures using only $(z+3)$ pairings. Individually verifying n signatures would cost $3n$ pairings.

We present a new signature scheme, CHP, derived from the Camenisch and Lysyanskaya signature scheme [58], which is secure in the random oracle model. CHP signatures require only one-third the space of the original CL signatures – on par with the shortest signatures known [37] –, but users may only issue one signature per period (e.g., users might only be allowed to sign one message per 300ms). We present a batch verifier for these signatures from many different signers that verifies n signatures using only three total pairings, instead of the $5n$ pairings required by n original CL signatures. Yet, our batch verifier has the restriction that it can only batch verify signatures made during the same period.

Often signatures and certificates need to be verified together. This happens implicitly in IBS schemes. To achieve this functionality with CHP signatures, we can issue signatures with CHP and certificates with the Boneh, Lynn, and Shacham signatures [37]. Then we can batch the CHP signatures (on any message from any signer) using a new batch verifier proposed herein; and we can batch the BLS certificates (on any public key from the same authority) using a known batch verifier that can batch verify n signatures from the *same* signer using only two pairings.

6.2.1 Efficiency of Prior Work and our Contributions

Efficiency will be given as an abstract cost for computing different functions. We begin by discussing prior work on RSA, DSA, and BLS signatures mostly for single signers, and then discuss our new work on the Waters variant, CHP and BLS signatures for many signers. Note that Lim [138] provides a number of efficient methods for doing m-term exponentiations and Granger and Smart [107] give improvements over the naive method for computing a product of pairings, which is why we state them explicitly.

m-$\mathsf{MultPairCost}^s_{\mathbb{G},\mathbb{H}}$	s m-term pairings $\prod_{i=1}^m \mathbf{e}(g_i, h_i)$ where $g_i \in \mathbb{G}$, $h_i \in \mathbb{H}$.
m-$\mathsf{MultExpCost}^s_{\mathbb{G}}(k)$	s m-term exponentiations $\prod_{i=1}^m g^{a_i}$ where $g \in \mathbb{G}$, $\lvert a_i\rvert = k$.
$\mathsf{PairCost}^s_{\mathbb{G},\mathbb{H}}$	s pairings $\mathbf{e}(g_i, h_i)$ for $i = 1 \ldots s$, where $g_i \in \mathbb{G}$, $h_i \in \mathbb{H}$.
$\mathsf{ExpCost}^s_{\mathbb{G}}(k)$	s exponentiations g^{a_i} for $i = 1 \ldots s$ where $g \in \mathbb{G}$, $\lvert a_i\rvert = k$.
$\mathsf{GroupTestCost}^s_{\mathbb{G}}$	Testing whether or not s elements are in the group $\mathbb{G}$.
$\mathsf{HashCost}^s_{\mathbb{G}}$	Hashing s values into the group $\mathbb{G}$.
$\mathsf{MultCost}^s$	s multiplications in one or more groups.

If $s = 1$ we will omit it. Throughout this chapter we assume that n is the number of message/signature pairs and ℓ_b is a security parameter such that the probability of accepting a batch that contains an invalid signature is at most $2^{-\ell_b}$.

RSA* is a modified version of RSA by Boyd and Pavlovski [40]. The difference to normal RSA is that the verification equation accepts a signature σ as valid if $\alpha\sigma^e = m$ for some element $\alpha \in \mathbb{Z}_m^*$ of order no more than 2, where m is the product of two primes. The signatures are usually between $1024 - 2048$ bits and the same for the public key. A single signer batch verifier for this signature scheme with cost $n\text{-}\mathsf{MultExpCost}^2_{\mathbb{Z}_m}(\ell_b) + \mathsf{ExpCost}_{\mathbb{Z}_m}(k)$, where k is the number of bits in the public exponent e, can be found in [40]. Note that verifying n signatures by verifying each signature individually only costs $\mathsf{ExpCost}^n_{\mathbb{Z}_m}(k)$, so for small values of e ($|e| < 2\ell_b/3$) the naive method is a faster way to verify RSA signatures and it can also handle signatures from multiple signers. Bellare et al. [23] present a screening algorithm for RSA that assumes distinct messages from the same signer and costs $2n + \mathsf{ExpCost}_{\mathbb{Z}_m}(k)$.

DSA** is a modified version of DSA from [150] compatible with the *small exponents test* from [40]. There are two differences to normal DSA. First there is no reduction modulo q, so the signatures are 672 bits instead of 320 bits and second, individual verification should check both a signature σ and $-\sigma$ and accept if one of them holds. Messages and public keys are both 160 bits long. Using the small exponents test the cost is $n\text{-}\mathsf{MultExpCost}_{\mathbb{G}}(\ell_b) + \mathsf{ExpCost}^2_{\mathbb{G}}(160) + \mathsf{HashCost}^n_{\mathbb{G}} + \mathsf{MultCost}^{2n+1}$ multiplications. This method works for a single signer only.

Waters is an IBS scheme derived from the HIBE scheme by Chatterjee and Sarkar [67] for which we provide a batch verifier without random oracles in Section 6.2.3. An interesting property of this scheme is that the identity does not need to be verified separately. Identities and messages are k bits divided into z logical chunks, each of k/z bits, where z is a security parameter, and a signature is three group elements. The computational effort required depends on the number of messages and the security parameters. Let $\mathsf{M} = n\text{-}\mathsf{MultExpCost}_{\mathbb{G}_T}(\ell_b) + n\text{-}\mathsf{MultExpCost}^3_{\mathbb{G}}(\ell_b) + \mathsf{PairCost}^3_{\mathbb{G},\mathbb{G}} + \mathsf{GroupTestCost}^{3n}_{\mathbb{G}} + \mathsf{MultCost}^3$ and refer to the table below for efficiency of the scheme.

$$\begin{array}{lll} n \leq 2z: & \mathsf{M} & +2n\text{-}\mathsf{MultPairCost}_{\mathbb{G},\mathbb{G}} + z\text{-}\mathsf{MultExpCost}^{2n}_{\mathbb{G}}(\frac{k}{z}) + \mathsf{ExpCost}^{2n}_{\mathbb{G}}(\ell_b) \\ n > 2z: & \mathsf{M} & +z\text{-}\mathsf{MultPairCost}_{\mathbb{G},\mathbb{G}} + \mathsf{ExpCost}^{2n}_{\mathbb{G}}(\frac{k}{z} + \ell_b) + \mathsf{MultCost}^{zn} \end{array}$$

The naive application of Waters to verify n signatures costs $\mathsf{PairCost}^{3n}_{\mathbb{G},\mathbb{G}} + z\text{-}\mathsf{MultExpCost}^{2n}_{\mathbb{G}}(\frac{k}{z}) + \mathsf{MultCost}^{4n}$. Also note that in many security applications we do not need to transmit the identity as a separate parameter, as it is already included in the larger protocol. For example, the identity may be the hardware address of the network interface card.

BLS is the signature scheme by Boneh et al. [37]. We discuss batch verifiers for BLS signatures based on the small exponents test. For a screening algorithm, aggregate signatures by Boneh, Gentry, Lynn and Shacham [35] can be used. The signature is only one group element in a bilinear group and the same for the public key. For different signers the cost of batch verification is $n\text{-}\mathsf{MultPairCost}_{\mathbb{G},\mathbb{G}} + n\text{-}\mathsf{MultExpCost}_{\mathbb{G}}(\ell_b) + \mathsf{PairCost}_{\mathbb{G},\mathbb{G}} + \mathsf{ExpCost}^n_{\mathbb{G}_T}(\ell_b) +$

$\mathsf{GroupTestCost}^n_{\mathbb{G}} + \mathsf{HashCost}^n_{\mathbb{G}}$, but for single signer it is $n\text{-}\mathsf{MultExpCost}^2_{\mathbb{G}}(\ell_b) + \mathsf{PairCost}^2_{\mathbb{G},\mathbb{G}} + \mathsf{GroupTestCost}^n_{\mathbb{G}} + \mathsf{HashCost}^n_{\mathbb{G}}$.

CHP is a new variant of Camenisch and Lysyanskaya signatures [58] presented in Section 6.2.4 designed specifically to enable efficient batch verification. The signature is only one bilinear group element and the same for the public key. Batch verification costs $n\text{-}\mathsf{MultExpCost}^2_{\mathbb{G}}(\ell_b) + n\text{-}\mathsf{MultExpCost}_{\mathbb{G}}(|w| + \ell_b) + \mathsf{PairCost}^3_{\mathbb{G},\mathbb{G}} + \mathsf{GroupTestCost}^n_{\mathbb{G}} + \mathsf{HashCost}^n_{\mathbb{G}}$, where w is the output of a hash function. However, the scheme has some additional restrictions.

Small Exponents and Bucket Tests. Recall the various testing techniques covered in Section 6.1.2. Our batch verifiers in this chapter make use of the small exponents test, but since the bucket test uses the small exponents test as a subroutine, we note that we can also use the bucket test to further speed up verification of many signatures.

6.2.2 Definitions

We now introduce some definitions, which we will use in the rest of this chapter.

6.2.2.1 Signature Schemes

Recall the definition of signature schemes from section 2.5. Now, we consider the case where we want to quickly verify a set of signatures on (possibly) different messages by (possibly) different signers. The input is $\{(t_1, m_1, \sigma_1), \ldots, (t_n, m_n, \sigma_n)\}$, where t_i specifies the verification key against which σ_i is purported to be a signature on message m_i. We extend the definitions of Bellare et al. [23] to deal with multiple signers. And this is an important point that was not a concern with only a single signer: *one or more of the signers may be maliciously colluding.*

Definition 6.1 (Batch Verification of Signatures) *Let τ be a security parameter. Suppose* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is a signature scheme, $k, n \in \mathrm{poly}(\tau)$, and $(pk_1, sk_1), \ldots, (pk_k, sk_k)$ are generated independently according to* $\mathsf{Gen}(1^\tau)$*. Let $PK = \{pk_1, \ldots, pk_k\}$. Then we call probabilistic* Batch *a batch verification algorithm when the following conditions hold:*

- *If $pk_{t_i} \in PK$ and* $\mathsf{Verify}(pk_{t_i}, m_i, \sigma_i) = 1$ *for all $i \in [1, n]$ then* $\mathsf{Batch}((pk_{t_1}, m_1, \sigma_1), \ldots, (pk_{t_n}, m_n, \sigma_n)) = 1$*.*
- *If $pk_{t_i} \in PK$ for all $i \in [1, n]$ and* $\mathsf{Verify}(pk_{t_i}, m_i, \sigma_i) = 0$ *for some $i \in [1, n]$, then* $\mathsf{Batch}((pk_{t_1}, m_1, \sigma_1), \ldots, (pk_{t_n}, m_n, \sigma_n)) = 0$ *except with probability negligible in τ, taken over the randomness of* Batch*.*

Note that Definition 6.1 requires that signing keys be generated honestly, but then they can be later held by an adversary. In practice, users could register their keys and prove some necessary properties of the keys at registration time [17].

Definition 6.2 (Screening of Signatures) *Let ℓ be the security parameter. Suppose* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is a signature scheme, $n \in \text{poly}(\ell)$ and $(pk_0, sk_0) \leftarrow \mathsf{Gen}(1^\ell)$. Let $\mathcal{O}_{sk_0}(\cdot)$ be an oracle that on input m outputs $\sigma = \mathsf{Sign}(sk_0, m)$. Then for all p.p.t. adversaries $\mathcal{A}$, we call probabilistic* Screen *a* screening algorithm *when $\mu(\ell)$ defined as follows is a negligible function:*

$$\begin{aligned}\Pr[(pk_0, sk_0) \leftarrow \mathsf{Gen}(1^\ell), (pk_1, sk_1) \leftarrow \mathsf{Gen}(1^\ell), \ldots, (pk_n, sk_n) \leftarrow \mathsf{Gen}(1^\ell),\\ D \leftarrow \mathcal{A}^{\mathcal{O}_{sk_0}(\cdot)}(pk_0, (pk_1, sk_1), \ldots, (pk_n, sk_n)) :\\ \mathsf{Screen}(D) = 1 \ \wedge \ (pk_0, m, \sigma) \in D \ \wedge \ m \notin Q] = \mu(\ell),\end{aligned}$$

where Q is the set of queries that $\mathcal{A}$ made to $\mathcal{O}_{sk_0}(\cdot)$ and for all $(pk_a, b, c) \in D$, $a \in \{0, \ldots, n\}$.

The above definition is generalized to the multiple-signer case from the single-signer screening definition of Bellare et al. [23].

Interestingly, screening is the (maximum) guarantee that most aggregate signatures offer if one were to attempt to batch verify a group of signatures by first aggregating them together and then executing the aggregate-verification algorithm. Consider the aggregate signature scheme of Boneh et al. [35] based on the BLS signatures [37]. First, we review the BLS signatures. Let $\mathsf{PSetup}(1^\tau) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$ To generate a key pair, choose a random $sk \in \mathbb{Z}_q$ and set $pk = g^{sk}$. A signature on message m is $\sigma = H(m)^{sk}$, where $H : \{0,1\}^* \rightarrow \mathbb{G}$ is a hash function. To verify signature σ on message m, one checks that $\mathbf{e}(\sigma, g) = \mathbf{e}(H(m), pk)$. Given a group of message-signature pairs $(m_1, \sigma_1), \ldots, (m_n, \sigma_n)$ (all purportedly from the same signer), BGLS aggregates them as $A = \prod_{i=1}^n \sigma_i$. Then all signatures can be verified in aggregate (i.e., screened) by testing that $\mathbf{e}(A, g) = \mathbf{e}(\prod_{i=1}^n H(m_i), pk)$. This scheme is *not*, however, a batch verification scheme since, for any $a \neq 1 \in \mathbb{G}$, the two *invalid* message-signature pairs $P_1 = (m_1, a \cdot H(m_1)^{sk})$ and $P_2 = (m_2, a^{-1} \cdot H(m_2)^{sk})$ will verify under Definition 6.2 (as BGLS prove [35]), but will not verify under Definition 6.1. Indeed, for some pervasive computing applications only guaranteeing screening would be disastrous, because only P_1 may be relevant information to forward to the next entity – and it will not verify once it arrives! Also recall the e-mail scenario from the introduction. If we only did screening on the server, a user could send n messages with invalid signatures (to different receivers) that would screen correctly. The sender could then later claim that he did not send one of the messages and indeed the signature will not verify unless one can get hold of *all* n messages! To be fair, batch verification is not what aggregate schemes were designed to do, but it is a common misuse of them.

Let us make one final observation about the relationship between batch verification and screening. Let $D = \{(t_1, m_1, \sigma_1), \ldots, (t_n, m_n, \sigma_n)\}$. We note that while $\mathsf{Screen}(D) = 1$ does *not* guarantee that $\mathsf{Verify}(pk_{t_i}, m_i, \sigma_i)$ for all i; it does guarantee that the holder of sk_{t_i} authenticated m_i. That is, for all i, the holder of sk_{t_i} helped to create σ_i, which may or may not be a valid signature for m_i. Thus, a screening scheme can be employed to hold users accountable for the messages they "sign" in a set D such that $\mathsf{Screen}(D) = 1$, but to do this the entire set D must be recorded or retransmitted to a third party. In the authenticated

email scenario, where the mail server is verifying the signatures on emails for many different users, releasing D (in the event of disputes) raises serious privacy issues. One could consider releasing a non-interactive zero-knowledge proof of knowledge of D such that $\mathsf{Screen}(D) = 1$, although the naive approach will require $O(|D|)$ space and $O(|D|)$ time to verify.

6.2.2.2 Batch Verifier

Let $\mathbb{G}$ be a group of prime order $q \in \Theta(2^\tau)$. Verification equations are represented by a *generic claim* X corresponding to a boolean relation of the following form: $\prod_{i=1}^{k} h_i^{c_i} \stackrel{?}{=} A$, for $k \in \text{poly}(\tau)$, $h_i \in \mathbb{G}$ and $c_i \in \mathbb{Z}_q^*$, for each $i = 1, \ldots, k$. A verifier Verify for a generic claim is a probabilistic poly(τ)-time algorithm which on input the representation $\langle A, h_1, \ldots, h_k, c_1, \ldots, c_k \rangle$ of a claim X, outputs *accept* if X holds and *reject* otherwise. Next definition describes a batch verifier.

Definition 6.3 (Batch Verifier) *Let $\mathbb{G}$ be a group of prime order $q \in \Theta(2^\tau)$. For each $j \in [1, \eta]$, where $\eta \in \text{poly}(\tau)$, let $X^{(j)}$ be a generic claim and let* Verify *be a verifier. We define* batch verifier *for* Verify *as a probabilistic poly(τ)-time algorithm which outputs* accept *if $X^{(j)}$ holds for all $j \in [1, \eta]$ whereas it outputs* reject *if $X^{(j)}$ does not hold for any $j \in [1, \eta]$ except with negligible probability.*

We now apply the small exponents test to a verification equation, and obtain a general batch verifier.

Theorem 6.1 *Let $\mathbb{G}$ be a group of prime order q, and let g be a generator of $\mathbb{G}$. For each $j \in [1, \eta]$, where $\eta \in \text{poly}(\tau)$, let $X^{(j)}$ corresponds to a generic claim as in Definition 6.3. For simplicity, assume that $X^{(j)}$ is of the form $A \stackrel{?}{=} Y^{(j)}$ where A is fixed for all j and all the input values to the claim $X^{(j)}$ are in the correct groups. For any random vector $\Delta = (\delta_1, \ldots, \delta_\eta)$ of ℓ_b bit elements from $\mathbb{Z}_q$, an algorithm* Batch *which tests the following equation $\prod_{j=1}^{\eta} A^{\delta_j} \stackrel{?}{=} \prod_{j=1}^{\eta} {Y^{(j)}}^{\delta_j}$ is a batch verifier that accepts an invalid batch with probability at most $2^{-\ell_b}$.*

Proof. The proof closely follows the proof of the small exponents test by Bellare et al. [23].

It is easy to see that if $A = Y^{(j)}$ holds for all $j \in [1, \eta]$, then $\prod_{j=1}^{\eta} A^{\delta_j} = \prod_{j=1}^{\eta} {Y^{(j)}}^{\delta_j}$ holds for any random vector $\Delta = (\delta_1, \ldots, \delta_\eta)$. We must now show the other direction, that if $A \neq Y^{(j)}$ for any $j \in [1, \eta]$, then Batch outputs *accept* only with probability $2^{-\ell_b}$. Since A and $Y^{(j)}$ are in $\mathbb{G}$, we can write $A = g^a$ and $Y^{(j)} = g^{y^{(j)}}$ for some $a, y^{(j)} \in \mathbb{Z}_q$. The batch verification equation can then be written as $\prod_{j=1}^{\eta} g^a = \prod_{j=1}^{\eta} g^{y^{(j)}} \Rightarrow g^{\sum_{j=1}^{\eta} a} = g^{\sum_{j=1}^{\eta} y^{(j)}}$. Now define $\beta_j = a - y^{(j)}$. Since Batch accepts it must be true that

$$\sum_{j=1}^{\eta} \beta_j \delta_j \equiv 0 \pmod{q} \tag{6.1}$$

Now assume that at least one of the individual equations do not hold. We assume without loss of generality that this is true for equation $j = 1$. This means that $\beta_1 \neq 0$. Since q is a prime then β_1 has an inverse γ_1 such that $\beta_1\gamma_1 \equiv 1 \pmod{q}$. This and Equation 6.1 gives us

$$\delta_1 \equiv -\gamma_1 \sum_{j=2}^{\eta} \delta_j\beta_j \pmod{q} \tag{6.2}$$

Let event E occurs if $A \neq Y^{(1)}$, but Batch accepts. Note that we do not make any assumptions about the remaining values. Let $\Delta' = \delta_2, \ldots, \delta_\eta$ denote the last $\eta - 1$ values of Δ and let $|\Delta'|$ be the number of possible values for this vector. Equation 6.2 says that given a fixed vector Δ' there is exactly one value of δ_1 that will make event E happen, or in other words that the probability of E given a randomly chosen δ_1 is $\Pr[E|\Delta'] = 2^{-\ell_b}$. So if we pick δ_1 at random and sum over all possible choices of Δ' we get $\Pr[E] \leq \sum(\Pr[E|\Delta'] \cdot \Pr[\Delta'])$. Plugging in the values, we get: $\Pr[E] \leq \sum_{i=1}^{2^{\ell_b(\eta-1)}} \left(2^{-\ell_b} \cdot 2^{-\ell_b(\eta-1)}\right) = 2^{-\ell_b}$. □

A natural question to ask is if this batch verifier also works for composite order groups. Unfortunately the answer is not straightforward. The reason for requiring a prime order group, is that for the proof of the small exponents test to go through, we need an element β_1 to have an inverse in $\mathbb{Z}_q$, which is the case if $gcd(\beta_1, q) = 1$. If q is prime this is always the case, but what if q is composite? If $q = p_1p_2$, where p_1, p_2 are primes, then this is the case except when β_1 is a multiple of p_1, p_2 or q. If β_1 is chosen at random it is very unlikely that an inverse does not exist, and the small exponents test will work in almost all cases. However, this really depends on the scheme, so if one wants to apply this method to a scheme set in a composite order group, one should examine the proof and make sure that it still applies to the chosen scheme.

6.2.3 Batch Verification without Random Oracles

In this section, we present a method for batch verifying an identity-based signature scheme Waters. This batch verification method can execute in different modes, optimizing for the lowest runtime. Let n be the number of certificate/signature pairs, let 2^k be the maximal number of users and assume that each message is k bits long. Let z be the additional security parameter required by the Waters scheme. Furthermore assume that the k bits are divided into z elements of k/z bits each. Then our batch verifier will verify n certificate/signature pairs with asymptotic complexity of the dominant operations roughly MIN$\{(2n + 3)\ ,\ (z + 3)\}$.

On the practical side, we note that as z grows there is a corresponding degradation in the concrete security of the IBS scheme (see [67] for a detailed discussion of these tradeoffs.), but on the other hand the efficiency of the scheme increases. Hence choosing a suitable value of z is a tradeoff. Setting $z = k/32$, however, seems a reasonable choice. Suppose we use SHA256 to hash all the messages ($k = 256$) and we choose the elements to be 32 bits ($k/z = 32$), then roughly when $n \geq 3$ batch verification becomes faster than individual

verification.

6.2.3.1 Batch Verification for Waters

We describe a batch verification algorithm for the Waters scheme [67], where the number of pairings depends on the security parameter and not on the number of signatures and where no random oracles are necessary. The underlying Waters signature scheme appears only implicitly in prior work, so let us clearly explain its origin. We begin with the observation by Boyen and Waters that an IBS scheme is realized by the key issuing algorithm of any (fully-secure) 2-level hierarchical identity-based encryption (HIBE) scheme [42].

In 2004, Boneh and Boyen described an efficient HIBE in the selective-ID security model [31]. In 2005, Waters described how to modify Boneh and Boyen identity-based encryption to make it fully-secure [189]. The difference between these two IBEs is the way the identity is evaluated. Assume the identity is a bit string $V = v_1 v_2 \dots v_m$. Instead of evaluating it as $u' g_1^V$ [31] then evaluating it as $u' \prod_{i=1}^{m} u_i^{v_i}$ [189] makes the scheme fully secure. In 2005 Naccache [149], and Chatterjee and Sarkar [66] independently showed how to generalize the Waters IBE to optimize it for efficiency. These ideas were extended in 2006 by Chatterjee and Sarkar to Waters HIBE and the resulting HIBE was proven secure in the standard model [67]. Finally the identity-based signature scheme implicitly defined by Chatterjee and Sarkar's HIBE, is what we will refer to as Waters.

The Waters scheme and its batch verification algorithm are both considerably more efficient than the non-generalized version. Indeed, the structure imposed by the generalization [67, 149] make the Waters scheme particularly well-suited for batch verification. We now explicitly describe the scheme we call Waters and then show how to batch verify these signatures.

We assume that the identities and messages are both bit strings of length k represented by z blocks of k/z bits each. (If this is not the case, then let k be the larger bit length and then pre-pad the shorter string with zeros.) Let $\mathsf{PSetup}(1^\tau) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$.

Setup First choose a secret $\alpha \in \mathbb{Z}_q$ and $h \in \mathbb{G}$ and calculate $A = \mathbf{e}(g,h)^\alpha$. Then pick two random integers $y_1', y_2' \in \mathbb{Z}_q$ and a random vector $y = (y_1, \dots, y_z) \in \mathbb{Z}_q^z$. The master secret key is $MK = h^\alpha$ and the public parameters are given as: $PP = g, A, u_1' = g^{y_1'}, u_2' = g^{y_2'}, u_1 = g^{y_1}, \dots, u_z = g^{y_z}$.

We use the notation of Chatterjee and Sarkar [67] to define the following function. Let $v = (v_1, \dots, v_z)$, where each v_i is a (k/z)-bit string. For $i \in \{1, 2\}$, let:

$$U_i(v) = u_i' \prod_{j=1}^{z} u_j^{v_j}.$$

Extract To create a private key for a user with identity $ID = \kappa_1, \dots, \kappa_z$, select $r \in \mathbb{Z}_q$ and return $K_{ID} = \left(h^\alpha \cdot U_1(ID)^r,\ g^{-r}\right)$.

Sign To sign a message $m = m_1, \ldots, m_z$ using private key $K = (K_1, K_2)$, select $s \in \mathbb{Z}_q$ and return

$$S = \left(K_1 \cdot U_2(m)^s,\ K_2,\ g^{-s}\right).$$

Verify To verify a signature $S = (S_1, S_2, S_3)$ from identity $ID = \kappa_1, \ldots, \kappa_z$ on message $m = m_1, \ldots, m_z$, check that:

$$A = \mathbf{e}(S_1, g) \cdot \mathbf{e}(S_2, U_1(ID)) \cdot \mathbf{e}(S_3, U_2(m)).$$

If this equation holds, output *accept*; otherwise output *reject*.

We now introduce a batch verifier for this signature scheme. The basic idea is to adopt the small exponents test from [23] and to take advantage of the peculiarities of pairings.

BatchVerify Suppose we want to batch verify n purported signatures. Let κ_j^i and m_j^i denote the j'th (k/z)-bit block of ID_i (the identity of the i'th signer) and M_i (the message signed by the i'th signer), respectively. Let $S^i = (S_1^i, S_2^i, S_3^i)$ denote the signature from the i'th signer. First check that all the identities have the correct length and that $S_1^i, S_2^i, S_3^i \in \mathbb{G}$ for all i. If not; output *reject*. Otherwise generate a vector $\Delta = (\delta_1, \ldots, \delta_n)$ where each δ_i is a random element of ℓ_b bits from $\mathbb{Z}_q$ and set

$$P = \mathbf{e}(\prod_{i=1}^{n} {S_1^i}^{\delta_i}, g) \cdot \mathbf{e}(\prod_{i=1}^{n} {S_2^i}^{\delta_i}, u_1') \cdot \mathbf{e}(\prod_{i=1}^{n} {S_3^i}^{\delta_i}, u_2').$$

Depending on the values of z and n (c.f. below), pick and check one of the following equations:

$$\prod_{i=1}^{n} A^{\delta_i} = P \cdot \prod_{i=1}^{n} \left(\mathbf{e}({S_2^i}^{\delta_i}, \prod_{j=1}^{z} u_j^{\kappa_j^i}) \cdot \mathbf{e}({S_3^i}^{\delta_i}, \prod_{j=1}^{z} u_j^{m_j^i}) \right) \tag{6.3}$$

$$\prod_{i=1}^{n} A^{\delta_i} = P \cdot \prod_{j=1}^{z} \mathbf{e}(\prod_{i=1}^{n} ({S_2^i}^{\kappa_j^i} \cdot {S_3^i}^{m_j^i})^{\delta_i}, u_j) \tag{6.4}$$

Output *accept* if the chosen equation holds; otherwise output *reject*.

Let us discuss which equation should be picked. If $n < 2z$, use equation 6.3; otherwise, use equation 6.4.

Theorem 6.2 *The above algorithm is a batch verifier for the Waters.*

Proof. Applying Theorem 6.1 to the verification equation for the Waters scheme, we see that the following is a batch verifier for the Waters scheme:

$$\begin{aligned}\prod_{i=1}^{n} A^{\delta_i} &= \prod_{i=1}^{n} \left(\mathbf{e}(S_1^i, g) \cdot \mathbf{e}(S_2^i, U_1(ID_i)) \cdot \mathbf{e}(S_3^i, U_1(M_i))\right)^{\delta_i} \\ &= \mathbf{e}(\prod_{i=1}^{n} {S_1^i}^{\delta_i}, g) \cdot \prod_{i=1}^{n} \mathbf{e}({S_2^i}^{\delta_i}, u_1' \prod_{j=1}^{z} u_j^{\kappa_j^i}) \cdot \prod_{i=1}^{n} \mathbf{e}({S_3^i}^{\delta_i}, u_2' \prod_{j=1}^{z} u_j^{m_j^i}) \\ &= P \cdot \prod_{i=1}^{n} \left(\mathbf{e}({S_2^i}^{\delta_i}, \prod_{j=1}^{z} u_j^{\kappa_j^i}) \cdot \mathbf{e}({S_3^i}^{\delta_i}, \prod_{j=1}^{z} u_j^{m_j^i}) \right) \end{aligned} \tag{6.5}$$

All we need now is to show that equation 6.3 is equivalent to equation 6.4. Since for all i, (S_1^i, S_2^i, S_3^i) are in the right group, we can write $S_2^i = g^{b_i}$ and $S_3^i = g^{c_i}$ for some elements $b_i, c_i \in \mathbb{Z}_q$. Now we rewrite the part inside the parenthesis of equation 6.3 and get equation 6.4:

$$\begin{aligned}&\prod_{i=1}^{n} \mathbf{e}({S_2^i}^{\delta_i}, \prod_{j=1}^{z} u_j^{\kappa_j^i}) \cdot \prod_{i=1}^{n} \mathbf{e}({S_3^i}^{\delta_i}, \prod_{j=1}^{z} u_j^{m_j^i}) \\ &\qquad = \prod_{i=1}^{n} \left(\mathbf{e}(g^{b_i}, g^{\sum_{j=1}^{z} \kappa_j^i y_j}) \cdot \mathbf{e}(g^{c_i}, g^{\sum_{j=1}^{z} m_j^i y_j}) \right)^{\delta_i} \\ &\qquad = \prod_{i=1}^{n} \left(\mathbf{e}(g, g)^{\sum_{j=1}^{z} (\delta_i b_i \kappa_j^i y_j + \delta_i c_i m_j^i y_j)} \right) \\ &\qquad = \prod_{j=1}^{z} \left(\mathbf{e}(g, g)^{y_j \sum_{i=1}^{n} (\delta_i b_i \kappa_j^i + \delta_i c_i m_j^i)} \right) \\ &\qquad = \prod_{j=1}^{z} \mathbf{e}(\prod_{i=1}^{n} ({S_2^i}^{\kappa_j^i} \cdot {S_3^i}^{m_j^i})^{\delta_i}, u_j). \end{aligned}$$

□

Efficiency Note. A Waters signature consists of three group elements, but since it is identity-based there is no public key, and we assume that the identity is given "for free" e.g. it could be the hardware address of the network interface card. Hence the size of the signature that verifies both the message and the identity depends only on the size of these group elements. We have described the scheme in the symmetric bilinear setting $\mathbf{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ because the original scheme does not work in the asymmetric bilinear setting $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. However, by switching the order of the elements in the first pairing and modifying the public parameters accordingly, the scheme also works in the asymmetric bilinear setting. However, security would be under the Decisional Co-Bilinear Diffie-Hellman assumption instead.

In the symmetric bilinear setting elements must be around 512 bits for security comparable to 1024 bits RSA, which gives us a total signature size of 1536 bits. In the asymmetric bilinear setting the elements S_2 and S_3 can be represented using 160 bits, whereas S_1 needs 512 bits. So all in all we can

represent the signature on the message and the identity using only 832 bits. However, it might not be efficient to test membership of the group $\mathbb{G}_2$, which is needed for batch verification.

6.2.4 Faster Batch Verification with Restrictions

In this section, we present a second method for batch verifying signatures together with their accompanying certificates. We propose using the BLS signature scheme [37] for the certificates and a modified version of the CL signature scheme [58] for signing messages. This method requires only two pairings to verify n certificates (from the same authority) and three pairings to verify n signatures (from possibly different signers). The cost for this significant efficiency gain is some usage restrictions, although as we will discuss, these restrictions may not be a problem for some of the applications we have in mind.

Certificates We use a batch verifier for BLS signatures from the same authority as described in Section 6.2.4.1. The scheme is secure under the CDH assumption in the random oracle model. Verifying n BLS signatures costs $n\text{-}\mathsf{MultExpCost}^2_{\mathbb{G}}(\ell_b) + \mathsf{PairCost}^2_{\mathbb{G},\mathbb{G}} + \mathsf{GroupTestCost}^n_{\mathbb{G}} + \mathsf{HashCost}^n_{\mathbb{G}}$, using the Section 6.2.1 notation.

Signatures We describe a new signature scheme CHP with a batch verifier in Section 6.2.4.2. The scheme is secure under the LRSW assumption in the plain model when the size of the message space is a polynomial and in the random oracle model when the size of the message space is super-polynomial. We assume that there are discrete time or location identifiers $\phi \in \Phi$. A user can issue at most one signature per ϕ (e.g., this might correspond to a device being allowed to broadcast at most one message every 300ms) and only signatures from the same ϕ can be batch verified together. To verify n CHP signatures, costs $n\text{-}\mathsf{MultExpCost}^2_{\mathbb{G}}(\ell_b) + n\text{-}\mathsf{MultExpCost}_{\mathbb{G}}(|w| + \ell_b) + \mathsf{PairCost}^3_{\mathbb{G},\mathbb{G}} + \mathsf{GroupTestCost}^n_{\mathbb{G}} + \mathsf{HashCost}^n_{\mathbb{G}}$, where w is the output of a hash function.

6.2.4.1 Batch Verification of BLS Signatures

We describe a batch verifier for *many signers* for the Boneh et al. signatures [37] described in Section 6.2.2, using the small exponents test [23].

Batch Verify: Given purported signatures σ_i from n users on *distinct* messages M_i for $i = 1 \ldots n$, first check that all public keys pk_i where $i \in [1, n]$ are valid, and that $\sigma_i \in \mathbb{G}$ for all i. If not; output *reject*. Otherwise compute $h_i = H(M_i)$ and generate a vector $\delta = (\delta_1, \ldots, \delta_n)$ where each δ_i is a random element of ℓ_b bits from $\mathbb{Z}_q$. Check that $\mathbf{e}(\prod_{i=1}^n \sigma_i^{\delta_i}, g) = \prod_{i=1}^n \mathbf{e}(h_i, pk_i)^{\delta_i}$. If this equation holds, output *accept*; otherwise output *reject*.

Theorem 6.3 *The algorithm above is a batch verifier for BLS signatures.*

Proof. We apply Theorem 6.1 to the verification equation for the BLS scheme, and conclude that this is indeed a batch verifier for BLS signatures:

$$\prod_{i=1}^{n} \mathbf{e}(\sigma_i, g)^{\delta_i} = \prod_{i=1}^{n} \mathbf{e}(h_i, pk_i)^{\delta_i} \Leftrightarrow \mathbf{e}(\prod_{i=1}^{n} \sigma_i^{\delta_i}, g) = \prod_{i=1}^{n} \mathbf{e}(h_i, pk_i)^{\delta_i} \tag{6.6}$$

□

Single Singer for BLS. However, BLS [37] previously observed that if we have a single signer with public key v, the verification equation can be written as $\mathbf{e}(\prod_{i=1}^{n} \sigma_i^{\delta_i}, g) = \mathbf{e}(\prod_{i=1}^{n} h_i^{\delta_i}, v)$ which reduces the load to only two pairings.

Theorem 6.4 ([37]) *The algorithm above is a single-signer, batch verifier for BLS signatures.*

6.2.4.2 A New Signature Scheme CHP

In this section we introduce a new signature scheme secure under the LRSW assumption [141], which is based on the Camenisch and Lysyanskaya signatures [58].

The Original CL Scheme. Recall the Camenisch and Lysyanskaya signature scheme [58]. Let $\mathsf{PSetup}(1^\tau) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$. Choose the secret key $sk = (x, y) \in \mathbb{Z}_q^2$ at random and set $X = g^x$ and $Y = g^y$. The public key is $pk = (X, Y)$. To sign a message $m \in \mathbb{Z}_q^*$, choose a random $a \in \mathbb{G}$ and compute $b = a^y$, $c = a^x b^{xm}$. Output the signature (a, b, c). To verify, check whether $\mathbf{e}(X, a) \cdot \mathbf{e}(X, b)^m = \mathbf{e}(g, c)$ and $\mathbf{e}(a, Y) = \mathbf{e}(g, b)$ holds.

CHP: A version of the CL Scheme Allowing Batch Verification. Our goal is to batch-verify CL signatures made by different signers. That is we need to consider how to verify equations of the form $\mathbf{e}(X, a) \cdot \mathbf{e}(X, b)^m = \mathbf{e}(g, c)$ and $\mathbf{e}(a, Y) = \mathbf{e}(g, b)$. The fact that the values X, a, b, and c are different for each signature seems to prevent efficient batch verification. Thus, we need to find a way such that many different signers share some of these values. Obviously, X and c need to be different. Now, depending on the application, all the signers can use the same value a by choosing a as the output of some hash function applied to, e.g., the current time period or location. We then note that all signers can use the same b in principle, i.e., have all of them share the same y as it is sufficient for each signer to hold only one secret value (i.e., $sk = x$). Indeed, the only reason that the signer needs to know y is to compute b. However, it turns out that if we define b such that $\log_a b$ is not known, the signature scheme is still secure. So, for instance, we can derive b in a similar way to a using a second hash function. Thus, all signers will virtually sign using the same y per time period (but a different one for each period).

We note that the idea of sharing some value between the signers in order to efficiently perform some operation on the signatures is not new. Gentry and Ramzan present an identity based aggregate signature scheme [100] in which signatures can only be aggregated if all signers agree on some dummy message that none of them have used before.

Let us now describe the resulting scheme. Let $\mathsf{PSetup}(1^\tau) \to (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$. Let $\phi \in \Phi$ denote the current time period or location, where $|\Phi|$ is polynomial. Let $\mathcal{M}$ be the message space, for now let $\mathcal{M} = \{0,1\}^*$. Let $H_1 : \Phi \to \mathbb{G}$, $H_2 : \Phi \to \mathbb{G}$, and $H_3 : \mathcal{M} \times \Phi \to \mathbb{Z}_q$ be different hash functions.

Gen Choose a random $x \in \mathbb{Z}_q$ and set $X = g^x$. Set $sk = x$ and $pk = X$.

Sign If this is the first call to Sign during period $\phi \in \Phi$, then on input message $m \in \mathcal{M}$, set $w = H_3(m||\phi)$, $a = H_1(\phi)$, $b = H_2(\phi)$ and output the signature $\sigma = a^x b^{xw}$. Otherwise, abort.

Verify On input message-period pair (m, ϕ) and purported signature σ, compute $w = H_3(m||\phi)$, $a = H_1(\phi)$ and $b = H_2(\phi)$, and check that $\mathbf{e}(\sigma, g) = \mathbf{e}(a, X) \cdot \mathbf{e}(b, X)^w$. If true, output *accept*; otherwise output *reject*.

Theorem 6.5 *Under the LRSW assumption in $\mathbb{G}$, the CHP signature scheme is existentially unforgeable in the random oracle model for message space $\mathcal{M} = \{0,1\}^*$.*

Proof. We show that if there exists a p.p.t. adversary $\mathcal{A}$ that succeeds with probability ϵ in forging CHP signatures, then we can construct a p.p.t. adversary $\mathcal{B}$ that solves the LRSW problem with probability $\epsilon \cdot |\Phi|^{-1} \cdot q_H^{-1}$ in the random oracle model, where q_H is the maximum number of oracle queries $\mathcal{A}$ makes to H_3 during any period $\phi \in \Phi$. Recall that $|\Phi|$ is a polynomial. Adversary $\mathcal{B}^{\mathcal{O}_{X,Y}(\cdot)}$ against LRSW operates as follows on input $(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e}, X, Y)$. Let τ be the security parameter. We assume that Φ is pre-defined. Let q_H be the maximum number of queries $\mathcal{A}$ makes to H_3 during any period $\phi \in \Phi$.

1. *Setup:* Send the parameters $(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$ to $\mathcal{A}$. Choose a random $w' \in \mathcal{M}$ and query $\mathcal{O}_{X,Y}(w')$ to obtain a LRSW instance (w', a', b', c'). Choose a random $\phi' \in \Phi$. Treat H_1, H_2, H_3 as random oracles. Allow $\mathcal{A}$ access to the hash functions H_1, H_2, H_3.

2. *Key Generation:* Set $pk^* = X$. Output to $\mathcal{A}$ the key pk^*.

3. *Oracle queries:* $\mathcal{B}$ responds to $\mathcal{A}$'s hash and signing queries as follows. Choose random r_i and s_i in $\mathbb{Z}_q$ for each time period (except ϕ'). Set up H_1 and H_2 such that:

$$H_1(\phi_i) = \begin{cases} g^{r_i} & \text{if } \phi_i \neq \phi' \\ a' & \text{otherwise} \end{cases} \tag{6.7}$$

and

$$H_2(\phi_i) = \begin{cases} g^{s_i} & \text{if } \phi_i \neq \phi' \\ b' & \text{otherwise} \end{cases} \tag{6.8}$$

Pick a random j in the range $[1, q_H]$. Choose random $t_{l,i} \in \mathbb{Z}_q$, such that $t_{l,i} \neq w'$, for $l \in [1, q_H]$ and $i \in [1, |\Phi|]$. Set up H_3 such that:

$$H_3(m_l||\phi_i) = \begin{cases} t_{l,i} & \text{if } \phi_i \neq \phi' \text{ or } l \neq j \\ w' & \text{otherwise} \end{cases} \tag{6.9}$$

$\mathcal{B}$ records $m^* := m_j$. Finally, set the signing query oracle such that on the lth query involving period ϕ_i:

$$\mathcal{O}_{sk^*}(m_l||\phi_i) = \begin{cases} \text{abort} & \text{if } \phi_i = \phi' \text{ and } l \neq j \\ c' & \text{else if } \phi_i = \phi' \text{ and } l = j \\ X^{r_i} X^{(s_i) t_{l,i}} & \text{otherwise} \end{cases} \tag{6.10}$$

4. *Output:* At some point $\mathcal{A}$ stops and outputs a purported forgery $\sigma \in \mathbb{G}$ for some (m_l, ϕ_i). If $\phi_i \neq \phi'$, $\mathcal{B}$ did not guess the correct period and thus $\mathcal{B}$ outputs a random guess for the LRSW game. If $m_l = m^*$ or the CHP signature does not verify, $\mathcal{A}$'s output is not a valid forgery and thus $\mathcal{B}$ outputs a random guess for the LRSW game. Otherwise, $\mathcal{B}$ outputs $(t_{l,i}, a', b', \sigma)$ as the solution to the LRSW game.

We now analyze $\mathcal{B}$'s success. If $\mathcal{B}$ is not forced to abort or issue a random guess, then we note that $\sigma = H_1(\phi_i)^x H_2(\phi_i)^{x \cdot H_3(m_l||\phi_i)}$. In this scenario $\phi_i = \phi'$ and $t_{l,i} \neq w'$. We can substitute as $\sigma = (a')^x (b')^{x \cdot (t_{l,i})}$. Thus, we see that $(t_{l,i}, a', b', \sigma)$ is indeed a valid LRSW instance. Thus, $\mathcal{B}$ succeeds at LRSW whenever $\mathcal{A}$ succeeds in forging CHP signatures, except when $\mathcal{B}$ is forced to abort or issue a random guess. First, when simulating the signing oracle, $\mathcal{B}$ is forced to abort whenever it incorrectly guesses which query to H_3, during period ϕ', $\mathcal{A}$ will eventually query to $\mathcal{O}_{sk^*}(\cdot, \cdot)$. Since all outputs of H_3 are independently random, $\mathcal{B}$ will be forced to abort with probability at most q_H^{-1}. Next, provided that $\mathcal{A}$ issued a valid forgery, then $\mathcal{B}$ is only forced to issue a random guess when it incorrectly guesses which period $\phi \in \Phi$ that $\mathcal{A}$ will choose to issue its forgery. Since, from the view of $\mathcal{A}$ conditioned on the event that $\mathcal{B}$ has not yet aborted, all outputs of the oracles are perfectly distributed as either random oracles (H_1, H_2, H_3) or as a valid CHP signer $(\mathcal{O}_{sk^*})$. Thus, this random guess is forced with probability at most $|\Phi|^{-1}$. Thus, if $\mathcal{A}$ succeeds with ϵ probability, then $\mathcal{B}$ succeeds with probability $\epsilon \cdot |\Phi|^{-1} \cdot q_H^{-1}$. □

On Removing the Random Oracles. In the previous proof, notice that we treated hash functions H_1, H_2 and H_3 as independent random oracles which were (statically) programmed in $|\Phi|$, $|\Phi|$, and $|\Phi| \cdot |\mathcal{M}|$ points, respectively, where Φ is the set of time period identifiers and $\mathcal{M}$ is the signing message space. Recall that, as before, $|\Phi|$ is restricted to be polynomial in the security parameter. Now, for sufficiently short message spaces, e.g., ISO defined error messages, we can replace all three random oracles in the security proof of CHP by concrete hash functions. Suppose that given a set of pairs $(x_1, y_1), \ldots, (x_k, y_k)$, it is possible to efficiently sample a function $H : \{0,1\}^\tau \to \mathbb{G}$ (where $k < 2\tau + 1$) from a $(2\tau + 1)$-independent function family $\mathcal{H}$ such that for each $H \in \mathcal{H}$, we have $H(x_i) = y_i$ for $i = 1$ to k. If such types of hash function families exist then we could simple constrain them exactly as we programmed our random oracles.

Fortunately, Canetti, Halevi, and Katz [61] describe a method for efficiently constructing such a hash function family which allows to map strings to bilinear

map elements (or to map strings to elements in another prime-order algebraic group such as $\mathbb{Z}_q$). Boneh and Boyen describe another such family [31]. Any family satisfying the constraints above will work for our purposes, where H_1 and H_2 map into bilinear group $\mathbb{G}$ and H_3 maps into $\mathbb{Z}_q$. The construction remains as before and the new security proof simply uses concrete functions with constraints mirroring the points (statically) programmed in the oracles.

Lemma 6.1 *Under the LRSW assumption in $\mathbb{G}$, the CHP signature scheme is existentially unforgeable in the plain model when $|\mathcal{M}|$ are polynomial in the security parameter.*

Batch Verification of CHP Signatures. Batch verification of n signatures $\sigma_1, \ldots, \sigma_n$ on messages $m_1, \ldots, m_n$ for the same period ϕ can be done as follows. (Recall that each signer can issue at most one signature per time period. Thus, these n signatures are all from different signers.) Assume that user i with public key X_i signed message m_i. Set $w_i = H(m_i||\phi)$. First check that all public keys X_i where $i \in [1, n]$ are valid, and that $\sigma_i \in \mathbb{G}$ for all i. If not; output *reject*. Otherwise pick a vector $\Delta = (\delta_i, \ldots, \delta_n)$ with each element being a random ℓ_b-bit number and check that $\mathbf{e}(\prod_{i=1}^n \sigma_i^{\delta_i}, g) = \mathbf{e}(a, \prod_{i=1}^n X_i^{\delta_i}) \cdot \mathbf{e}(b, \prod_{i=1}^n X_i^{w_i \delta_i})$. If this equation holds, output *accept*; otherwise output *reject*.

Theorem 6.6 *The algorithm above is a batch verifier for CHP signatures.*

Proof. Applying Theorem 6.1 to the verification equation for the CHP scheme, we get that this is indeed a batch verifier for CHP signatures.

$$\prod_{i=1}^{n} \mathbf{e}(\sigma_i, g)^{\delta_i} = \prod_{i=1}^{n} (\mathbf{e}(a, X_i) \cdot \mathbf{e}(b, X_i)^{w_i})^{\delta_i} = \prod_{i=1}^{n} \mathbf{e}(a, X_i)^{\delta_i} \cdot \prod_{i=1}^{n} \mathbf{e}(b, X_i)^{w_i \delta_i} \tag{6.11}$$

$$\Leftrightarrow \mathbf{e}(\prod_{i=1}^{n} \sigma_i^{\delta_i}, g) = \mathbf{e}(a, \prod_{i=1}^{n} X_i^{\delta_i}) \cdot \mathbf{e}(b, \prod_{i=1}^{n} X_i^{w_i \delta_i})$$

□

CHP Without Batch Verification. So far we have described CHP only as an efficient signature scheme to batch verify, but for completeness we note that if we are not interested in batch verification, CHP is still a fairly efficient regular signature scheme without any restrictions.

Gen Choose a random $x \in \mathbb{Z}_q$ and set $X = g^x$. Set $sk = x$ and $pk = X$.

Sign Generate a value $\phi \in \Phi$ that has never been used by the signer before. Then on input message $m \in \mathcal{M}$, set $w = H_3(m||\phi)$, $a = H_1(\phi)$, $b = H_2(\phi)$, $\sigma = a^x b^{xw}$ and output the signature (σ, ϕ).

Verify On input message m and purported signature (σ, ϕ), compute $w = H_3(m||\phi)$, $a = H_1(\phi)$ and $b = H_2(\phi)$, and check that $\mathbf{e}(\sigma, g) = \mathbf{e}(ab^w, X)$. If true, output *accept*; otherwise output *reject*.

This is very similar to the original scheme. Note that the only change is that ϕ is now generated independently from all other signers and included as part of the signature, which makes the scheme unsuitable for batch verification (since the probability that many signers will share the same value of ϕ is small). However, now that we are only interested in individual verification, we can rewrite the original verification equation $\mathbf{e}(\sigma, g) = \mathbf{e}(a, X) \cdot \mathbf{e}(b, X)^w$ as $\mathbf{e}(\sigma, g) = \mathbf{e}(ab^w, X)$ which requires only two pairings to verify. Finally note that this variant of the verification equation does not depend on how ϕ was generated, and can always be used for individual verification if needed.

Efficiency Note. First, we observe that the CHP signatures are *very* short, requiring only one element in $\mathbb{G}$. Since the BLS signatures also require only one element in $\mathbb{G}$, and since a public key for the CHP scheme is also only one group element, the entire signature plus certificate could be transmitted in three $\mathbb{G}$ elements. In order to get the shortest representation for these elements, we need to use asymmetric bilinear maps $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where $\mathbb{G}_1 \neq \mathbb{G}_2$, which will allow elements in $\mathbb{G}_1$ to be 160 bits and elements of $\mathbb{G}_2$ to be 512 bits for a security level comparable to RSA-1024. For CHP signatures we need to hash into $\mathbb{G}_1$ which according to Galbraith, Paterson and Smart can be done efficiently [96]. To summarize; using BLS and CHP we can represent the signature plus certificate using approximately 832 bits with security comparable to RSA-1024, compared to approximately 3072 bits for actually using RSA-1024.

Second, suppose one uses the universal one-way hash functions described by Canetti et al. [61] to remove the random oracles from CHP. These hash functions require one exponentiation per constraint. In our case, we may require as many as $|\Phi| \cdot |\mathcal{M}|$ constraints. Thus, the cost to compute the hashes may dampen the efficiency gains of batch verification. However, our scheme will benefit from improvements in the construction of universal one-way hash functions with constraints.

If CHP is used as a signatures scheme without an efficient batch verifier, the signature require one group element in $\mathbb{G}$ and one element in Φ where the size of Φ only needs to be large enough to represent the number of times a user might want to sign with the same private key. Verification of a single CHP signature requires two pairings.

6.3 Practical Short Signature Batch Verification

In the previous section we attempted to speed up the verification of short signatures, by showing how to *batch verify* two short pairing-based signature schemes, so that the total number of dominant (pairing) operations was independent of the number of signatures to verify. However, there are still several unanswered questions, which we will examine in this section:

In the previous section, efficiency was stated as a rather abstract measure. Furthermore, to reduce the total number of pairings, we had to add additional operations, such as random number generation and small modular exponentiations, so it is unclear exactly how well the scheme will actually perform in practice.

Second, the existing theoretical literature contains many good ideas on batch verification, but these ideas were scattered across multiple papers, and it was not always clear how to safely employ these techniques from scheme to scheme. In Section 6.3.1.1, we present a general framework for how to securely batch verify a set of pairing-based equations.

Third, we present a detailed study of when and how our framework can be applied to existing regular, identity-based, group, ring, and aggregate signature schemes in Section 6.3.2. To our knowledge, these are the *first* known results for batch verification of group and ring signatures. This is particularly exciting, because it is the first step towards making short, *privacy-friendly* authentication fast enough for deployment in real systems.

Finally we have not yet address the practical issue of what to do if batch verification fails. How does one detect which signatures in the batch are invalid? Does this detection process eliminate all of the efficiency gains of batch verification? Fortunately, our empirical studies reveal good news: Invalid signatures can be detected via a recursive divide-and-conquer approach, and if $\leq 10\%$ of the signatures are invalid, then batch verification is still more efficient than individual verification. At the time we conducted these experiments, the divide-and-conquer approach was the best method known to us thus this is the only method studied in our section 6.3.3.2. Recently, Law and Matt proposed three new techniques for finding invalid pairing-based signatures in a batch [134]. In particular, one of their techniques, which is the most efficient for large batch sizes, allows to save approximately half the time needed by the divide-and-conquer approach. Instead, the other two proposed methods appear to be more suitable to identify invalid signatures in small sized batches and with a low number of bad signatures.

Overall, the conclusion of this section is that many interesting short signatures can be batch verified, and that batch verification is a valuable tool for system implementors.

6.3.1 A Framework for Pairing-Based Batch Verification

We now provide some useful observations to determine when pairing equations can be batch verified. Theorem 6.1 tells us how to generate a secure batch verifier from any number of pairing-based claims, but there is no guarantee that the resulting equation will be any more efficient than just verifying each claim independently. An efficient batch verifier (if possible) comes from optimizing the equation we get after applying Theorem 6.1.

6.3.1.1 Techniques to Speed Up Batch Verification

Armed with Theorem 6.1, let us back up for a moment to get a complete picture of how to develop an efficient batch verifier. Immediately following the summary, we will explain the details.

Framework Summary: Suppose you have η bilinear equations, to batch verify them, do the following:

1. Apply Technique 1 to the individual verification equation, if applicable.
2. Apply Theorem 6.1 to the equations, this involves checking membership in the expected algebraic groups and using the small exponents test.
3. Optimize the resulting equation using Techniques 2, 3 and 4.
4. If batch verification fails, use the divide-and-conquer approach to identity the bad signatures.

Technique 1 *Change the verification equation.* Recall that a Σ-protocol is a three step protocol (commit, challenge, response) allowing a prover to prove various statements to a verifier. Using the Fiat-Shamir heuristic [93] any Σ-protocol can be turned into a signature scheme, by forming the challenge as the hash of the commitment and the message to be signed. The signature is then either (commit, response) or (challenge, response). The latter is often preferred, since the challenge is usually smaller than the commitment, which results in a smaller signature. However, we observed that this often causes batch verification to become very inefficient, whereas using (commit, response) results in a much more suitable verification equation.

We use this technique to help batch the Hess IBS scheme [114] and the group signatures of Boneh, Boyen, and Shacham [33]. Indeed, we believe that prior attempts to batch verify group signatures overlooked this idea and thus came up without efficient solutions.

Combination Step: Given η pairing-based claims, apply Theorem 6.1 to obtain a single equation. The combination step actually consist of two substeps:

1. *Check Membership*: Check that all elements are in the correct subgroup. Only elements that could be generated by an adversary needs to be checked (e.g., elements of a signature one wants to verify). Public parameters need not be checked, or could be checked once and for all.
2. *Small Exponents Test*: Combine all equations into one and apply the small exponents test.

Next, optimize this single equation using any of the following techniques in any order.

Technique 2 *Move the exponent around.* When a pairing of the form $\mathbf{e}(g_i, h_i)^{\delta_i}$ appears, move the exponent δ_i into $\mathbf{e}()$. Since elements of $\mathbb{G}$ are usually smaller than elements of $\mathbb{G}_T$, this gives a small speedup when computing the exponentiation.

$$\text{Replace } \mathbf{e}(g_i, h_i)^{\delta_i} \text{ with } \mathbf{e}(g_i^{\delta_i}, h_i)$$

Remember that it is also possible to move an exponent out of the pairing, or move it between the two elements of the pairing. In some instances, this allows for further optimizations depending on the implementation of the pairing.

Technique 3 *When two pairings with a common first or second element appear, they can be combined.* A simple example could be the following:

$$\text{Replace } \mathbf{e}(a, g) \cdot \mathbf{e}(b, g) \text{ with } \mathbf{e}(ab, g)$$

When applying the batching technique from Theorem 6.1 to verify η equations, one will often end up with an equation that can be optimized using this technique. It will work like this:

$$\text{Replace } \prod_{i=1}^{\eta} \mathbf{e}(g_i^{\delta_i}, h) \text{ with } \mathbf{e}(\prod_{i=1}^{\eta} g_i^{\delta_i}, h)$$

When batching η instances using Theorem 6.1 this will reduce η pairings to one. This is also worth keeping in mind when designing schemes, or picking schemes that one wants to batch verify. Pick a scheme so that when $\mathbf{e}(g, h)$ appears in the verification equation, g or h is fixed.

In rare cases it might even be useful to apply this technique "in reverse", e.g. splitting a single pairing into two pairings, to allow for more efficient batch verification. An example is the ring signature scheme by Boyen [41] where this is needed to apply Technique 4 below.

Technique 4 *Waters hash.* In his IBE, Waters described how to hash identities to values in $\mathbb{G}_1$ [189], using a technique that was subsequently employed in several signature schemes. Assume the identity is a bit string $V = v_1 v_2 \dots v_m$, then given public parameters $u', u_1, \dots, u_m \in \mathbb{G}_1$, the hash is $u' \prod_{i=1}^{m} u_i^{v_i}$. Following works by Naccache [149] and Chatterjee and Sarkar [66, 67] documented the generalization where instead of evaluating the identity bit by bit, divide the k bit identity bit string into z blocks, and use the Waters hash as before. (In Section 6.3.3, we SHA1 hash our messages to a 160-bit string, and use $z = 5$ as proposed in [149].) In Section 6.2.3.1 we pointed out the following method for faster batching of Waters hashes.

$$\text{Replace } \prod_{j=1}^{\eta} \mathbf{e}(g_j, \prod_{i=1}^{m} u_i^{v_{ij}}) \text{ with } \prod_{i=1}^{m} \mathbf{e}(\prod_{j=1}^{\eta} g_j{}^{v_{ij}}, u_i)$$

In this section, we apply this technique to schemes with structures related to the Waters hash; namely, the ring signatures of Boyen [41] and the aggregate signatures of Lu et al. [139].

Handling Invalid Signatures. If there is even a single invalid signature in the batch, then the batch verifier will reject with high probability, but in many real world situations a signature collection may contain invalid signatures caused by accidental data corruption, or possibly malicious activity by an adversary seeking to degrade service. In many cases, the ratio of invalid signatures to valid could be quite small, and yet a standard batch verifier will reject the entire collection.

In some cases this may not be a serious concern. For example, sensor networks with a high level of redundancy may choose to simply drop messages that cannot be efficiently verified. Alternatively, systems may be able to cache and/or individually verify important messages when batch verification fails. However, in some applications, it might be critical to tolerate some percentage of invalid signatures without losing the performance advantage of batch verification.

In this section we employ a recursive *divide-and-conquer* approach, similar to that of Pastuszak, Pieprzyk, Michalek, and Seberry [159], as: First, shuffle the incoming batch of signatures, and if batch verification fails, simply divide the collection into two halves, and recurse on the halves. When this process terminates, the batch verifier outputs the index of each invalid signature. Through careful implementation and caching of intermediate results, much of the work of the batch verification (i.e., computing the product of many signature elements) can be performed once over the full signature collection, and need not be repeated when verifying each sub-collection. Thus, the cost of each recursion is dominated by the number of pairings used in the batch verification algorithm. In Section 6.3.3.2 we show that even if up to 10% of the signatures are invalid, this technique can still be faster than individual verification.

6.3.2 Applying the Framework to Signature Schemes

Now we apply our framework to a (non-exhaustive) collection of existing regular, identity-based, group, ring, and aggregate signature schemes. After a careful literature search, we are presenting only the schemes with the best results (although we often make a note in the particular sections about common schemes that do not seem to batch well.) To our knowledge, our batch verifiers for the group and ring signatures are the first proposals for batching privacy-friendly authentication. Figure 6.1 shows a summary of our results.

Scheme	Model	Individual-Verify	Batch-Verify	Reference
Group Signatures				
BBS [33]	RO	5η	2	§ 6.3.2.1
ID-based Ring Signatures				
CYH [75]	RO	2η	2	§ 6.3.2.2
Ring Signatures				
Boyen [41] (same ring)	plain	$\ell \cdot (\eta + 1)$	$\min\{\eta \cdot \ell + 1, 3 \cdot \ell + 1\}$	§ 6.3.2.2
Signatures				
BLS [37]	RO	2η	$s + 1$	[37]
CHP [54] (time restrictions)	RO	3η	3	6.2.4.2
ID-based Signatures				
Hess [114]	RO	2η	2	§ 6.3.2.3
ChCh [65]	RO	2η	2	[134]
Waters [42, 67, 149, 189]	plain	3η	$\min\{(2\eta + 3), (z + 3)\}$	6.2.3.1
Aggregate Signatures				
BGLS [35] (same users)	RO	$\eta(\ell + 1)$	$\ell + 1$	§ 6.3.2.4
Sh [176] (same users)	RO	$\eta(\ell + 2)$	$\ell + 2$	§ 6.3.2.4
LOSSW [139] (same sequence)	plain	$\eta(\ell + 1)$	$\min\{(\eta + 2), (\ell \cdot k + 3)\}$	§ 6.3.2.4

Figure 6.1: **Summary of signatures schemes for which our framework applies.** Let η be the number of signatures to verify, s be the number of distinct signers involved and ℓ be either the size of a ring or the size of an aggregate. Boyen batch verifier requires each signature to be issued according to the same ring. Aggregate verifiers work for signatures related to the same set of users. In CHP, only signatures from the same time period can be batched and z is a (small) parameter (e.g., 8). In LOSSW, k is the message bit-length. RO stands for random oracle.

6.3.2.1 Short Group Signatures

In this section, we show how to modify the short group signatures of Boneh, Boyen, and Shacham (BBS) [33] in order to allow for a batch verifier which requires only 2 pairings at the expense of an increase in the signature size. Fortunately, however, this increase in size still keeps the signatures shorter than their corresponding RSA-based counterparts. To our knowledge, these are the first known results for batch verification of group signatures.

Recall that a group signature scheme allows any member to sign on behalf of the group in such a way that anyone can verify a signature using the group public key while nobody, but the group manager, can identify the actual signer. A group signature scheme consists in four algorithm: Gen, Sign, Verify and Open, that, respectively generate public and private keys for users and the group manager, sign a message on behalf of a group, verify the signature on a message according to the group and trace a signature to a signer.

The BBS Group Signatures. Let $\mathsf{PSetup}(1^\tau) \rightarrow (q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbf{e})$, where $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$ is a hash function. Let ℓ be the number of users in a group. Note that the BBS scheme requires a computable isomorphism $\psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$ since their definition of the SDH assumption is based on it, but

unfortunately such an isomorphism does exist for all pairings. Fortunately, Boneh and Boyen have proposed a cleaner definition which doesn't require said isomorphism [32].

Gen Select a generator $g_2 \in \mathbb{G}_2$ at random and set $g_1 \leftarrow \psi(g_2)$. Select $h \xleftarrow{\$} \mathbb{G}_1 \setminus \{1_{\mathbb{G}_1}\}$, $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_q^*$, and set u, v such that $u^{r_1} = v^{r_2} = h$. Select $\gamma \xleftarrow{\$} \mathbb{Z}_q^*$, and set $w = g_2^{\gamma}$. For each $i = 1, \ldots, n$, select $x_i \xleftarrow{\$} \mathbb{Z}_q^*$, and set $f_i \leftarrow g_1^{\frac{1}{\gamma + x_i}}$. The public key is $\mathsf{gpk} = (g_1, g_2, h, u, v, w)$, the group manager's secret key is $\mathsf{gmsk} = (r_1, r_2)$ and the secret key of the i'th user is $\mathsf{gsk[i]} = (f_i, x_i)$.

Sign Given a group public key $\mathsf{gpk} = (g_1, g_2, h, u, v, w)$, a user private key (f, x) and a message $M \in \{0,1\}^*$, compute the signature σ as follows: (1) Select $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$ and compute $T_1 \leftarrow u^{\alpha}; T_2 \leftarrow v^{\beta}; T_3 \leftarrow f \cdot h^{\alpha+\beta}$. (2) Compute $\gamma_1 \leftarrow x \cdot \alpha$ and $\gamma_2 \leftarrow x \cdot \beta$. (3) Select $r_\alpha, r_\beta, r_x, r_{\gamma_1}, r_{\gamma_2} \xleftarrow{\$} \mathbb{Z}_q$ and compute $R_1 \leftarrow u^{r_\alpha}$; $R_2 \leftarrow v^{r_\beta}$; $R_3 \leftarrow \mathbf{e}(T_3, g_2)^{r_x} \cdot \mathbf{e}(h, w)^{-r_\alpha - r_\beta} \cdot \mathbf{e}(h, g_2)^{-r_{\gamma_1} - r_{\gamma_2}}$; $R_4 \leftarrow T_1^{r_x} \cdot u^{-r_{\gamma_1}}$; $R_5 \leftarrow T_2^{r_x} \cdot v^{-r_{\gamma_2}}$. (4) Compute $c \leftarrow H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$. (5) Compute $s_\alpha \leftarrow r_\alpha + c \cdot \alpha$; $s_\beta \leftarrow r_\beta + c \cdot \beta$; $s_x \leftarrow r_x + c \cdot x$; $s_{\gamma_1} \leftarrow r_{\gamma_1} + c \cdot \gamma_1$; $s_{\gamma_2} \leftarrow r_{\gamma_2} + c \cdot \gamma_2$. The signature is $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\gamma_1}, s_{\gamma_2})$.

Verify Given a group public key $\mathsf{gpk} = (g_1, g_2, h, u, v, w)$, a message M and a group signature $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\gamma_1}, s_{\gamma_2})$, compute the values $R_1 \leftarrow u^{s_\alpha} \cdot T_1^{-c}$; $R_2 \leftarrow v^{s_\beta} \cdot T_2^{-c}$; $R_3 \leftarrow \mathbf{e}(T_3, g_2)^{s_x} \cdot \mathbf{e}(h, w)^{-s_\alpha - s_\beta} \cdot \mathbf{e}(h, g_2)^{-s_{\gamma_1} - s_{\gamma_2}} \cdot \left(\mathbf{e}(T_3, w) \cdot \mathbf{e}(g_1, g_2)^{-1}\right)^c$;
$R_4 \leftarrow T_1^{s_x} \cdot u^{-s_{\gamma_1}}$; $R_5 \leftarrow T_2^{s_x} \cdot v^{-s_{\gamma_2}}$.

Accept if and only if $c \stackrel{?}{=} H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$.

An Efficient Batch Verifier for the BBS Group Signature Scheme. Computing R_3 is the most expensive part of the verification above, but at first glance it is not clear that this can be batched, because each R_3 is individually hashed. However, as described by Technique 1, the signature and the verification algorithm can be modified in order to efficiently apply our framework at the expense of an increase in the signature size.

Let $\sigma = (T_1, T_2, T_3, R_3, c, s_\alpha, s_\beta, s_x, s_{\gamma_1}, s_{\gamma_2})$ be the new signature, and then change the verification as follows:

NewVerify Given a group public key $\mathsf{gpk} = (g_1, g_2, h, u, v, w)$, a message M and a group signature $\sigma = (T_1, T_2, T_3, R_3, c, s_\alpha, s_\beta, s_x, s_{\gamma_1}, s_{\gamma_2})$, compute the values $R_1 \leftarrow u^{s_\alpha} \cdot T_1^{-c}$; $R_2 \leftarrow v^{s_\beta} \cdot T_2^{-c}$; $R_4 \leftarrow T_1^{s_x} \cdot u^{-s_{\gamma_1}}$; $R_5 \leftarrow T_2^{s_x} \cdot v^{-s_{\gamma_2}}$, then check the following pairing based equation

$$\mathbf{e}(T_3, g_2)^{s_x} \cdot \mathbf{e}(h, w)^{-s_\alpha - s_\beta} \cdot \mathbf{e}(h, g_2)^{-s_{\gamma_1} - s_{\gamma_2}} \cdot \left(\mathbf{e}(T_3, w) \cdot \mathbf{e}(g_1, g_2)^{-1}\right)^c \stackrel{?}{=} R_3. \tag{6.12}$$

Finally check if $c \stackrel{?}{=} H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$. Accept if all checks succeed and reject otherwise.

Now we are ready to define a batch verifier for η BBS purported group signatures, where the main objective is to cut down on the number of pairings required.

BBSBatchVerify Let $\mathsf{gpk} = (g_1, g_2, h, u, v, w)$ be the group public key, and let $\sigma_j = (T_{j,1}, T_{j,2}, T_{j,3}, R_{j,3}, c_j, s_{j,\alpha}, s_{j,\beta}, s_{j,x}, s_{j,\gamma_1}, s_{j,\gamma_2})$ be the j'th signature on the message M_j, for each $j = 1, \ldots, \eta$. For each $j = 1, \ldots, \eta$, compute the following values:

$$R_{j,1} \leftarrow u^{s_{j,\alpha}} \cdot T_{j,1}^{-c_j} \qquad R_{j,2} \leftarrow v^{s_{j,\beta}} \cdot T_{j,2}^{-c_j}$$
$$R_{j,4} \leftarrow T_{j,1}^{s_{j,x}} \cdot u^{-s_{j,\gamma_1}} \qquad R_{j,5} \leftarrow T_{j,2}^{s_{j,x}} \cdot v^{-s_{j,\gamma_2}}$$

Now for each $j = 1, \ldots, \eta$, check the following:

$$c_j \stackrel{?}{=} H(M_j, T_{j,1}, T_{j,2}, T_{j,3}, R_{j,1}, R_{j,2}, R_{j,3}, R_{j,4}, R_{j,5})$$

Then check the following *single* pairing based equation

$$\mathbf{e}(\prod_{j=1}^{\eta}(T_{j,3}^{s_{j,x}} \cdot h^{-s_{j,\gamma_1}-s_{j,\gamma_2}} \cdot g_1^{-c_j})^{\delta_j}, g_2) \cdot \mathbf{e}(\prod_{j=1}^{\eta}(h^{-s_{j,\alpha}-s_{j,\beta}} \cdot T_3^{c})^{\delta_j}, w) \stackrel{?}{=} \prod_{j=1}^{\eta} R_{j,3}^{\delta_j}. \tag{6.13}$$

where $(\delta_1, \ldots, \delta_\eta)$ is a random vector of ℓ_b bit elements from $\mathbb{Z}_q$. Accept if and only if all checks succeed.

Theorem 6.7 *For security level ℓ_b, the above algorithm is a batch verifier for the BBS group signature scheme, where the probability of accepting an invalid signature is $2^{-\ell_b}$.*

We show how to apply Theorem 6.1 in the following proof of Theorem 6.7.
Proofsketch. Let $\mathsf{gpk} = (g_1, g_2, h, u, v, w)$ be the group public key, and let $\sigma_j = (T_{j,1}, T_{j,2}, T_{j,3}, R_{j,3}, c, s_{j,\alpha}, s_{j,\beta}, s_{j,x}, s_{j,\gamma_1}, s_{j,\gamma_2})$ be the j'th signature on the message M_j, for each $j = 1, \ldots, \eta$. Since the BBS Batch Verify algorithm performs the same tests as the New Verify algorithm for each signature separately, we just need to prove that equation 6.13 is a batch verifier for the pairing based equation 6.12. From Theorem 6.1, for any random vector $(\delta_1, \ldots, \delta_\eta)$ of ℓ_b bit elements from $\mathbb{Z}_q$, the following pairing based equation

$$\prod_{j=1}^{\eta}(\mathbf{e}(T_{j,3}, g_2)^{s_{j,x}} \cdot \mathbf{e}(h, w)^{-s_{j,\alpha}-s_{j,\beta}} \cdot \mathbf{e}(h, g_2)^{-s_{j,\gamma_1}-s_{j,\gamma_2}}$$
$$\cdot \big(\mathbf{e}(T_{j,3}, w) \cdot \mathbf{e}(g_1, g_2)^{-1}\big)^{c_j})^{\delta_j} \stackrel{?}{=} \prod_{j=1}^{\eta} R_{j,3}^{\delta_j} \tag{6.14}$$

is a batch verifier for the pairing based equation 6.12. It is easy to see that equation 6.14 is equivalent to equation 6.13. Indeed, equation 6.13 is an optimized version of equation 6.14 obtained by applying techniques 2 and 3. □

Performance and Signature Length. The BBS batch verifier is suitable to verify many signatures issued by many group members on different messages. The original BBS signature consists of three elements of $\mathbb{G}_1$ and six elements of $\mathbb{Z}_q$ while its modified version, needed to construct the BBS batch verifier, requires three elements of $\mathbb{G}_1$, one element of $\mathbb{G}_T$, and six elements of $\mathbb{Z}_q$. When implemented in the 170-bit MNT curve proposed by Boneh et al., this results in a signature representation of approximately 2553 bits with security approximately equivalent to 1024-bit RSA. This is still shorter than the comparable (non-pairing) scheme of Ateniese, Camenisch, Joye, and Tsudik [10] which achieves a similar security level at a cost of at least 3872 bits.

6.3.2.2 Ring and Identity-based Ring Signatures

In this section, we show how to batch verify:

- The standard model ring signatures of Boyen (Boyen) [41] with the restriction that we can only batch ring signatures which have the same ring of ℓ signers using $\leq 3\ell + 1$ pairings.
- The random oracle model, identity-based ring signatures of Chow, Yiu, and Hui (CYH) [75], where even rings of different sizes involving different ring members on different messages, can be batched using only 2 pairings.

Recall that a ring signature scheme allows a signer to sign a message on behalf of a set of users which include the signer itself in such a way that a verifier is convinced that the signer is one of the ring members, but he cannot tell which member is the actual signer. A ring signature is a triple of algorithms Gen, Sign and Verify, that, respectively generate public and private keys for a user, sign a message on behalf of the ring and verify the signature on a message according to the ring. In an identity-based ring signature, a user can choose an arbitrary string, for example her email address, as her public key. The corresponding private key is then created by binding such a string which represents the user's identity with the master key of a trusted party called private key generator (PKG). Such a scheme consists of four algorithms: Setup, Gen, Sign and Verify. During Setup, the PKG sets the system parameters P_{pub} and chooses a master secret key msk. During Gen, the PKG gives the user a secret key based on her identity string. Then the signing and verification algorithms and verification algorithms operate as before, except that only P_{pub} and the ring members identities are needed in place of their public keys.

The CYH batch verifier processes a bunch of signatures issued by many different rings on many different messages by performing only two pairing evaluations. The CYH scheme is proven to be secure in the random oracle model. Boyen's batch verifier processes signatures issued by the same ring members. Indeed, by using techniques 3 and 4, the pairing $\mathbf{e}(S_i, \hat{A}_i \cdot \hat{B}_i^M \cdot \hat{C}_i^{t_i})$ can be split in the following product $\mathbf{e}(S_i, \hat{A}_i) \cdot \mathbf{e}(S_i^M, \hat{B}_i) \cdot \mathbf{e}(S_i^{t_i}, \hat{C}_i)$. Since variables $\hat{A}_i$, $\hat{B}_i$ and $\hat{C}_i$ assume the same value for signatures issued according to the same ring, it is easy to see that the Boyen's batch verifier of Figure 6.3 holds.

Figures 6.2 and 6.3 summarizes the scheme we consider and how to batch them, respectively.[1] The CYH scheme is fairly straightforward to batch, while the Boyen scheme required more creativity, especially in the application of techniques 3 and 4.

Scheme	Setup Key Generation	Signature	Verify
CYH	$(q, g_1, g_2, \mathbb{G}_1,$ $\mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{PSetup}(1^\tau)$ $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$ $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ $\alpha \xleftarrow{\$} \mathbb{Z}_q^*$ $msk \leftarrow \alpha$ $P_{pub} \leftarrow g^\alpha$ $sk \leftarrow H_1(ID)^\alpha$ $pk \leftarrow H_1(ID)$	Let $L = \{ID_1, ID_2, \ldots, ID_\ell\}$ Let ID_s be the signer $\forall i \in [1,\ell]$ s.t. $i \neq s$ $u_i \xleftarrow{\$} \mathbb{G}_1$ $h_i \leftarrow H_2(M\|L\|u_i)$ $r \xleftarrow{\$} \mathbb{Z}_q$ $u_s \leftarrow pk_s^r \cdot \left(\prod_{i \neq s} u_i \cdot pk_i^{h_i}\right)^{-1}$ $h_s \leftarrow H_2(M\|L\|u_s)$ $S = sk_s^{h_s + r}$ $\sigma \leftarrow (u_1, \ldots, u_\ell, S)$	Let $\sigma = (u_1, \ldots, u_\ell, S)$ $\forall i \in [1,\ell]$ $h_i \leftarrow H_2(M\|L\|u_i)$ $\mathbf{e}(\prod_{i=1}^{\ell} u_i \cdot pk_i^{h_i}, P_{pub}) \stackrel{?}{=} \mathbf{e}(S, g_2)$
Boyen	$(q, g_1, g_2, \mathbb{G}_1,$ $\mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{PSetup}(1^\tau)$ $H : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ $\psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$ $\hat{A}_0, \hat{B}_0, \hat{C}_0 \xleftarrow{\$} \mathbb{G}_2$ $a, b, c \xleftarrow{\$} \mathbb{Z}_q^*$ $A \leftarrow g_1^a$; $B \leftarrow g_1^b$; $C \leftarrow g_1^c$ $\hat{A} \leftarrow g_2^a$; $\hat{B} \leftarrow g_2^b$; $\hat{C} \leftarrow g_2^c$ $sk \leftarrow (a, b, c)$ $pk \leftarrow (A, B, C, \hat{A}, \hat{B}, \hat{C})$	Let $L = \{pk_1, pk_2, \ldots, pk_\ell\}$ where $pk_i = (A_i, B_i, C_i, \hat{A}_i, \hat{B}_i, \hat{C}_i)$ W.l.o.g., let pk_ℓ be the signer $s_0, s_1, \ldots, s_{\ell-1}, t_0, t_1, \ldots, t_\ell \xleftarrow{\$} \mathbb{Z}_q$ $\forall i \in [0, \ell-1]$, $S_i \leftarrow g_1^{s_i}$ $d \leftarrow \frac{1}{a_\ell + b_\ell \cdot M + c_\ell \cdot t_\ell}$ $S_\ell \leftarrow \left(g \cdot \prod_{i=0}^{\ell-1} (A_i \cdot B_i^M \cdot C_i^{t_i})^{-s_i}\right)^d$ $\sigma = (S_0, \ldots, S_\ell, t_0, \ldots, t_\ell)$	Let $\sigma = (S_0, \ldots, S_\ell, t_0, \ldots, t_\ell)$ Let $D = \mathbf{e}(g_1, g_2)$ $\prod_{i=0}^{\ell} \mathbf{e}(S_i, \hat{A}_i \cdot \hat{B}_i^M \cdot \hat{C}_i^{t_i}) \stackrel{?}{=} D$

Figure 6.2: **Ring signature schemes that we consider.** We denote by P_{pub}, sk and pk the system parameters, user private key and user public key, respectively. Moreover, we denote with pk_i and sk_i the public and private keys of the i-th user in the ring. A ring signature on a message M is denoted by σ and ℓ represents the ring size.

6.3.2.3 Signature and Identity-based Signature Schemes

In this section, we review the known batch verifiers for [37]:

- The short, random oracle model signatures of Boneh et al. (BLS) [37] for signatures by the *same* signer, which require 2 pairings to batch. (In Section 6.3.3, we'll use this scheme to batch certificates.)
- The short, random oracle model signatures from Section 6.2.4.2 for signatures by different signers within the *same* time period, which require 3 pairings to batch.

[1] In the course of the study about schemes suitable to apply our framework, we noticed that the identity-based ring signature scheme proposed in [11] is a very nice candidate. Unfortunately, we found that, for ring size greater than two, the security proof has a flaw. After hearing of this proof flaw, Brent Waters translated it into an attack on the scheme (personal communication). It is still open to see if such a scheme is indeed secure for rings of size two.

Scheme	Batch Verification Precomputation / Batch Verification Equation	Techniques
CYH	Let $\sigma_j = (u_{j,1}, \ldots, u_{j,\ell}, S_j)$ and $L_j = \{ID_{j,1}, \ldots, ID_{j,\ell_j}\}$; $\forall i, j\ h_{j,i} \leftarrow H_2(M_j \| L_j \| u_{j,i})$ / $\mathbf{e}(\prod_{j=1}^{\eta}\prod_{i=1}^{\ell_j} pk_{j,i}^{(h_{j,i}+u_{j,i})\cdot\delta_j}, P_{pub})$	2,3
Boyen	Let $\sigma_j = (S_{j,0}, \ldots, S_{j,\ell}, t_{j,0}, \ldots, t_{j,\ell})$, $pk_i \leftarrow (A_i, B_i, C_i, \hat{A}_i, \hat{B}_i, \hat{C}_i)$ and $D = \mathbf{e}(g_1, g_2)$ / If $\eta < 3$, $\prod_{j=1}^{\eta}\prod_{i=0}^{\ell} \mathbf{e}(S_{j,i}^{\delta_j}, \hat{A}_i \cdot \hat{B}_i^{m_{j,i}} \cdot \hat{C}_i^{t_{j,i}}) = \prod_{j=1}^{\eta} D^{\delta_j}$ / Otherwise, $\prod_{i=0}^{\ell}\left(\mathbf{e}(\prod_{j=1}^{\eta} S_{j,i}^{\delta_j}, \hat{A}_i)\cdot\mathbf{e}(\prod_{j=1}^{\eta} S_{j,i}^{\delta_j m_{j,i}}, \hat{B}_i)\cdot\mathbf{e}(\prod_{j=1}^{\eta} S_{j,i}^{\delta_j t_{j,i}}, \hat{C}_i)\right) = \prod_{j=1}^{\eta} D^{\delta_j}$	2,3,4

Figure 6.3: **Batch verifier for the ring signature and ID-based ring signature schemes we consider.** Let η be the number of signatures to verify and M_j be the message corresponding to the j'th signature σ_j. With $pk_{j,i}$ and ℓ_j we denote the public key of the i'th ring member and the size of the ring associated to the j'th signature, respectively. The vector $(\delta_1, \ldots, \delta_\eta)$ in $\mathbb{Z}_q$ is required by the small exponents test.

- The standard model, identity-based signatures, called Waters, from Section 6.2.3.1, which were implicitly defined by Waters [189] and then generalized by subsequent works [42, 67, 149]. These signatures can be batched using $\leq z+3$ pairings, where z is a small security parameter (e.g., $z = 5$.)

We then present new results on batch verifiers, requiring only two pairings, for:

- The random oracle model, identity-based signatures of Cha and Cheon (ChCh) [65].
- The random oracle model, identity-based signatures of Hess (Hess) [114].

Interestingly, the identity-based signature due to Sakai, Ohgishi, and Kasahara [171] is very similar to those above, and yet its subtle differences make it a poor candidate for batching.

The BLS signature scheme is presented in Section 6.2.4.1. As also noticed by the authors, BLS is suitable to verify a bunch of purported signatures either issued from the same signer on different messages or by different public keys on the same message in a faster way than simply verifying each signature separately. Indeed, consider η BLS signatures $\sigma_1, \ldots, \sigma_\eta$ issued by means of the BLS signature algorithm (see Figure 6.4) under the same public key pk on different messages $M_1 \ldots, M_\eta$. According to the BLS verification equation (see Figure 6.4), 2η pairing evaluations are needed to verify each equation separately, while applying techniques 2 and 3, only two pairing evaluations suffice: $\mathbf{e}(\prod_{j=1}^{\eta} H(M_j)^{\delta_j}, pk) = \mathbf{e}(\prod_{j=1}^{\eta} \sigma_j^{\delta_j}, g_2)$, for some vector $(\delta_1, \ldots, \delta_\eta)$ in $\mathbb{Z}_q$.

A similar approach can be used to batch verify signatures issued by the same public key with only two pairing evaluations. In Figure 6.5 we describe a more general batch verification equation, where we consider a bunch of signatures issued by s different signers. Applying technique 3, it is easy to see that the pairings on the left hand side of the BLS verification equation corresponding to

Scheme	Setup / Key Generation	Signature	Verification Precomputation / Verification Equation
BLS	$(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{PSetup}(1^\tau)$ $H : \{0,1\}^* \rightarrow \mathbb{G}$ $\alpha \xleftarrow{\$} \mathbb{Z}_q$ $sk \leftarrow \alpha$; $pk \leftarrow g_2^\alpha$	$\sigma \leftarrow H(M)^{sk}$	$\mathbf{e}(H(M), pk) \stackrel{?}{=} \mathbf{e}(\sigma, g_2)$
CHP	Let Φ be the set of time periods. $(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{PSetup}(1^\tau)$ $H_1 : \Phi \rightarrow \mathbb{G}_1$, $H_2 : \Phi \rightarrow \mathbb{G}_1$ $H_3 : \{0,1\}^* \times \Phi \rightarrow \mathbb{Z}_q$ $\alpha \xleftarrow{\$} \mathbb{Z}_q$ $sk \leftarrow \alpha$; $pk \leftarrow g_2^\alpha$	$a \leftarrow H_1(\phi)$ $h \leftarrow H_2(\phi)$ $b \leftarrow H_3(M\|\|\phi)$ $\sigma \leftarrow a^{sk} \cdot h^{sk \cdot b}$	$a \leftarrow H_1(\phi); h \leftarrow H_2(\phi); b \leftarrow H_3(M\|\|\phi)$ $\mathbf{e}(\sigma, g_2) \stackrel{?}{=} \mathbf{e}(a, pk) \cdot \mathbf{e}(h, pk)^b$

Figure 6.4: **Signature Schemes that we consider.** We denote by pk and sk the public key and the private key of a user, respectively. We denote by σ a signature on a message M. In CHP, ϕ is a time period in the set of time periods Φ.

Scheme	Batch Verification Precomputation / Batch Verification Equation	Techniques
BLS	$\prod_{i=1}^{s} \mathbf{e}(\prod_{\ell=i_1}^{i_{n_i}} H(M_\ell)^{\delta_\ell}, pk_i) \stackrel{?}{=} \mathbf{e}(\prod_{j=1}^{\eta} \sigma_j^{\delta_j}, g_2)$	2,3
CHP	$a \leftarrow H_1(\phi); h \leftarrow H_2(\phi); \forall j \in [1, \eta],\ b_j \leftarrow H_3(M_j\|\|\phi)$ $\mathbf{e}(\prod_{j=1}^{\eta} \sigma_j^{\delta_j}, g_2) \stackrel{?}{=} \mathbf{e}(a, \prod_{j=1}^{\eta} pk_j^{\delta_j}) \cdot \mathbf{e}(h, \prod_{j=1}^{\eta} pk_j^{b_j \cdot \delta_j})$	2,3

Figure 6.5: **Batch verifiers for the signature schemes we consider.** Let η be the number of signatures to verify. With pk_j we denote the public key of the user who issued the j'th signature. The vector $(\delta_1, \ldots, \delta_\eta)$ in $\mathbb{Z}_q$ is required by the small exponents test. In BLS, s is the number of different signer and n_i is the number of signatures issued by the i'th signer (for details see the text). In CHP, ϕ is a time period in the set of time periods Φ.

signatures issued by the same public key can be grouped in a single pairing. This yields to the BLS batch verification equation of Figure 6.5, where each signer i, for $i = 1, \ldots, s$, is responsible of the n_i out of the η signatures identified by the indices $i_1, \ldots, i_{n_i}$. The BLS batch verification equation of Figure 6.5 requires $s + 1$ pairing evaluations. A similar approach can be used to quickly batch verify signatures on m different messages with $m + 1$ pairing evaluations.

The CL based scheme CHP from Section 6.2.4.2 (see Figure 6.4) allows efficient batch verification of signatures made by different signers provided that all signatures have been issued during the same period of time. Since the values g_2, a and h are the same for all signatures, from techniques 2 and 3, the CHP batch verification equation shown in Figure 6.5 requires only three pairings, instead of the 5η pairings required to verify η original CL signatures.

Scheme	Setup Key Generation	Sign	Verification Precomputation Verification Equation
ChCh	$(q, g_1, g_2, \mathbb{G}_1,$ $\mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{PSetup}(1^\tau)$ $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$ $H_2 : \{0,1\}^* \times \mathbb{G}_1 \rightarrow \mathbb{Z}_q$ $\alpha \xleftarrow{\$} \mathbb{Z}_q$ $msk \leftarrow \alpha$; $P_{pub} \leftarrow g_2^\alpha$ $sk \leftarrow H_1(ID)^\alpha$; $pk \leftarrow H_1(ID)$	$s \xleftarrow{\$} \mathbb{Z}_q$ $S_1 \leftarrow pk^s$ $a \leftarrow H_2(M\|\|S_1)$ $S_2 \leftarrow sk^{s+a}$ $\sigma \leftarrow (S_1, S_2)$	Let $\sigma = (S_1, S_2)$, $a \leftarrow H_2(M\|\|S_1)$ $\mathbf{e}(S_2, g_2) \stackrel{?}{=} \mathbf{e}(S_1 \cdot pk^a, P_{pub})$
Hess	$(q, g_1, g_2, \mathbb{G}_1,$ $\mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{PSetup}(1^\tau)$ $H_1 : \{0,1\}^* \rightarrow \mathbb{G}$ $H_2 : \{0,1\}^* \times \mathbb{G}_T \rightarrow \mathbb{Z}_q$ $\alpha \xleftarrow{\$} \mathbb{Z}_q$ $msk \leftarrow \alpha$; $P_{pub} \leftarrow g_2^\alpha$ $sk \leftarrow H_1(ID)^\alpha$; $pk \leftarrow H_1(ID)$	$h \xleftarrow{\$} \mathbb{G}$ $s \xleftarrow{\$} \mathbb{Z}_q$ $S_1 \leftarrow \mathbf{e}(h, g_2)^s$ $a \leftarrow H_2(M\|\|S_1)$ $S_2 \leftarrow sk^a \cdot h^s$ $\sigma \leftarrow (S_1, S_2)$	Let $\sigma = (S_1, S_2)$, $a \leftarrow H_2(M\|\|S_1)$ $\mathbf{e}(S_2, g_2) \stackrel{?}{=} \mathbf{e}(pk, P_{pub})^a \cdot S_1$
Waters	$(q, g_1, g_2, \mathbb{G}_1,$ $\mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{PSetup}(1^\tau)$ $\alpha \xleftarrow{\$} \mathbb{Z}_q$; $h \xleftarrow{\$} \mathbb{G}_1$ $A \leftarrow \mathbf{e}(h, g_2)^\alpha$ $y'_1, y'_2, y_1, y_2, \ldots, y_z \xleftarrow{\$} \mathbb{Z}_q$ $u'_1 \leftarrow g_1^{y'_1}$; $u'_2 \leftarrow g_1^{y'_2}$ $\forall \ell \in [1, z]$, $u_\ell \leftarrow g_1^{y_\ell}$ $\hat{u_1}' \leftarrow g_2^{y'_1}$; $\hat{u_2}' \leftarrow g_2^{y'_2}$ $\forall \ell \in [1, z]$, $\hat{u_\ell} \leftarrow g_2^{y_\ell}$ $msk \leftarrow h^\alpha$ $P_{pub} \leftarrow (A, u'_1, u'_2, u_1, \ldots, u_z,$ $\hat{u_1}', \hat{u_2}', \hat{u_1}, \ldots, \hat{u_z})$ $r \xleftarrow{\$} \mathbb{Z}_q$ $k_1 \leftarrow h^\alpha \cdot (u'_1 \cdot \prod_{i=1}^z u_i^{\kappa_i})^r$ $k_2 \leftarrow g_1^{-r}$ $sk \leftarrow (k_1, k_2)$	$s \xleftarrow{\$} \mathbb{Z}_q$ $S_1 \leftarrow k_1 \cdot (u'_2 \cdot \prod_{i=1}^z u_i^{m_i})^s$ $S_2 \leftarrow k_2$ $S_3 \leftarrow g_1^{-s}$ $\sigma \leftarrow (S_1, S_2, S_3)$	Let $\sigma = (S_1, S_2, S_3)$ and $A = \mathbf{e}(h, g_2)^\alpha$ $\mathbf{e}(S_1, g_2) \cdot \mathbf{e}(S_2, \hat{u_1}' \cdot \prod_{i=1}^z \hat{u_i}^{\kappa_i}) \cdot \mathbf{e}(S_3, \hat{u_2}' \cdot \prod_{i=1}^z \hat{u_i}^{m_i}) \stackrel{?}{=} A$

Figure 6.6: **Identity-based signature schemes that we consider.** We denote by msk, P_{pub}, sk and pk the master key, the system parameters, user private key and user public key, respectively. We denote by σ a signature on a message M. In Waters, z is the number of ℓ-bit chunks. Moreover, the identity ID and the message M are parsed as $\kappa_1, \ldots, \kappa_z$ and $m_1, \ldots, m_z$, respectively.

In the following we focus on batch verification for identity-based signature schemes. An identity based signature scheme consists of four algorithms: Setup, Key Generation, Sign and Verify. The public key generator PKG initializes the system during the Setup phase by choosing the system parameters P_{pub} which are made public. Moreover, the PKG chooses a master key msk and keeps it secret. The master key is used in the key generation phase along with the identity of a user to compute the user's private key. A user can sign a message by using the Sign algorithm. Finally, a verifier can check a signature on a message by using the Verify algorithm on input the signature, the public parameters and the identity of the signer. In Figure 6.6 we summarize the identity-based signature schemes we consider.

As shown in Figure 6.7, techniques 2 and 3 allow to construct a batch verifier which requires only two pairing evaluations for the schemes ChCh and Hess. Both ChCh and Hess schemes are proved secure in the random oracle model. The ChCh batch verifier of Figure 6.7 was also shown in [134]. Finally Waters is the identity-based signature scheme secure in the standard model, derived

Scheme	Batch Verification Precomputation Batch Verification Equation	Techniques
ChCh	Let $\sigma_j = (S_{j,1}, S_{j,2})$. $\forall j \in [1,\eta]$, $a_j \leftarrow H_2(M_j \| S_{j,1})$ $\mathbf{e}(\prod_{j=1}^{\eta} S_{j,2}^{\delta_j}, g_2) \stackrel{?}{=} \mathbf{e}(\prod_{j=1}^{\eta}(S_{j,1} \cdot pk_j^{a_j})^{\delta_j}, P_{pub})$	2,3
Hess	Let $\sigma_j = (S_{j,1}, S_{j,2})$. $\forall j \in [1,\eta]$, $a_j \leftarrow H_2(M_j \| S_{j,1})$ $\mathbf{e}(\prod_{j=1}^{\eta} S_{j,2}^{\delta_j}, g_2) \stackrel{?}{=} \mathbf{e}(\prod_{j=1}^{\eta} pk_j^{a_j \cdot \delta_j}, P_{pub}) \cdot \prod_{j=1}^{\eta} S_{j,1}^{\delta_j}$	2,3
Waters	Let $\sigma_j = (S_{j,1}, S_{j,2}, S_{j,3})$ and $P_{pub} = (A, u_1', u_2', u_1, \ldots, u_z, \hat{u_1}', \hat{u_2}', \hat{u_1}, \ldots, \hat{u_z})$ If $z > 2\eta - 2$, $\mathbf{e}(\prod_{j=1}^{\eta} S_{j,1}, g_2) \cdot \prod_{j=1}^{\eta} \left(\mathbf{e}(S_{j,1}^{\delta_j}, \hat{u_1}' \prod_{i=1}^{z} \hat{u_j}^{k_{j,i}}) \cdot \mathbf{e}(S_{j,3}^{\delta_j}, \hat{u_2}' \prod_{i=1}^{z} \hat{u_j}^{m_{j,i}}) \right) \stackrel{?}{=} A^{\sum_{j=1}^{\eta} \delta_j}$ Otherwise, $\mathbf{e}(\prod_{j=1}^{\eta} S_{j,1}^{\delta_j}, g_2) \cdot \mathbf{e}(\prod_{j=1}^{\eta} S_{j,2}^{\delta_j}, \hat{u_1}') \cdot \mathbf{e}(\prod_{j=1}^{\eta} S_{j,3}^{\delta_j}, \hat{u_2}') \cdot \prod_{i=1}^{z} \mathbf{e}(\prod_{j=1}^{\eta}(S_{j,2}^{k_{j,i}} \cdot S_{j,3}^{m_{j,i}})^{\delta_j}, \hat{u_i}) \stackrel{?}{=} A^{\sum_{j=1}^{\eta} \delta_j}$	2,3, 4

Figure 6.7: **Batch verifiers for the id-based signature schemes we consider.** Let η be the number of signatures to verify. With pk_j we denote the public key of the user who issued the j'th signature σ_j on message M_j. The vector $(\delta_1, \ldots, \delta_\eta)$ in $\mathbb{Z}_q$ is required by the small exponents test. In Waters, z is the number of ℓ-bit chunks. Moreover, the identity ID_j and the message M_j corresponding to the j-th signature are parsed as $\kappa_{j,1}, \ldots, \kappa_{j,z}$ and $m_{j,1}, \ldots, m_{j,z}$, respectively.

from a number of contributions [67, 149, 189], as described in Section 6.2.3.1. As remarked in Figure 6.7, by using techniques 2, 3 and 4, Waters allows us to define a batch verifier where the number of pairing evaluations is proportional to the minimum between the number of signatures η and the number of chunks z.

6.3.2.4 Aggregate Signatures

Aggregate signatures were introduced by Boneh et al. [36]. An aggregate signature is a shorter representation of n signatures provided by different users on different messages. In particular, consider n signatures $\sigma_1, \ldots, \sigma_n$ on messages $M_1, \ldots, M_n$ issued by n users with public keys $pk_1, \ldots, pk_n$. An aggregate signature scheme provides an aggregation algorithm, which can be run by anyone and outputs a compressed short signature σ on input all σ_i, for $i = 1, \ldots, n$. Moreover, there is a verification algorithm that on inputs the signature σ the public keys $pk_1, \ldots, pk_n$ and the messages $M_1, \ldots, M_n$ decides if σ is a valid aggregate signature. Figure 6.8 reviews the aggregate signatures we consider. Sh scheme [176] requires the existence of a third party named *aggregator* who is responsible of aggregating signatures. LOSSW scheme [139], proved to be secure in the standard model, is a sequential aggregate signature scheme. The aggregate signature must be constructed sequentially, with each signer adding its signature in turn. Figure 6.9 shows the corresponding batch verifier obtained by using our framework. Following the line of Theorem 6.7 it is easy to see that the pairing based equations in Figure 6.9 are batch verifiers for the corresponding schemes when all aggregate signatures are issued by the same set

of users.

In this section, we show batch verifiers for the aggregate signature scheme by Boneh, Gentry, Lynn and Shacham (BGLS) [35] (same users) and Shao (Shao) [176] (same users), and for the sequential aggregate scheme by Lu, Ostrovsky, Sahai, Shacham and Waters (LOSSW) [139] (same sequence).

Scheme	Setup Key Generation	Aggregate Signature	Verification
BGLS	Same as BLS Same as BLS	Let σ_i be a BLS signature on message M_i under private key pk_i $\sigma \leftarrow \prod_{i=1}^{\ell} \sigma_i$	$\mathbf{e}(\sigma, g_2) \stackrel{?}{=} \prod_{i=1}^{\ell} \mathbf{e}(H(M_i), pk_i)$
Sh	Same as BLS For users and aggregator, same as BLS	Let σ_i be a BLS signature on message M_i under private key pk_i If all BLS signatures are valid, the aggregator use its secret key sk_{ag} to compute $\sigma \leftarrow H(M_1 \| \ldots \| M_\ell)^{sk_{ag}} \cdot \prod_{i=1}^{\ell} \sigma_i$	$\mathbf{e}(\sigma, g_2) \stackrel{?}{=} \mathbf{e}(H(M_1,\ldots, M_\ell), pk_{ag}) \cdot \prod_{i=1}^{\ell} \mathbf{e}(H(M_i), pk_i)$
LOSSW	$(q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathsf{PSetup}(1^\tau)$ $\alpha, y' \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ $(y_1, \ldots, y_k) \stackrel{\$}{\leftarrow} \mathbb{Z}_q^k$ $\mathbf{y} = (y_1, \ldots, y_k)$ $u' \leftarrow g_1^{y'}$ $\hat{u}' \leftarrow g_2^{y'}$ $\forall i = 1, \ldots, k,$ $u_i \leftarrow g_1^{y_i}$ $\hat{u}_i \leftarrow g_2^{y_i}$ $\mathbf{u} = (u_1, \ldots, u_k,$ $\hat{u}_1, \ldots, \hat{u}_k)$ $A \leftarrow \mathbf{e}(g_1, g_2)^\alpha$ $sk \leftarrow (\alpha, y', \mathbf{y})$ $pk \leftarrow (A, u', \hat{u}', \mathbf{u})$	Let $\sigma' = (\prod_{i}^{\ell-1} g_1^{\alpha_i} \cdot \prod_{i=1}^{\ell-1} (u_i' \prod_{t=1}^{k} u_{i,t}^{m_{i,t}})^{r'}$, $g_1^{r'}) = (S_1', S_2')$ be an aggregate so far on a set of messages $\{M_1, \ldots, M_{\ell-1}\}$ under public keys $\{pk_1, \ldots, pk_{\ell-1}\}$. Let M_ℓ be the message to sign under public key pk_ℓ and corresponding secret key sk_ℓ. We denote $pk_i = (A_i, u_i', \hat{u_i}', u_{i,1}, \ldots, u_{i,k}, \hat{u_{i,1}}, \ldots, \hat{u_{i,k}})$, $sk_i = (\alpha_i, y_i', y_{i,1}, \ldots, y_{i,k})$ and $M_i = m_{i,1}, \ldots, m_{i,k}$. $w_1 \leftarrow S_1' \cdot g_1^{\alpha} \cdot (S_2')^{(y_\ell' + \sum_{t=1}^{k} y_{\ell,t} \cdot m_{\ell,t})}$ $w_2 \leftarrow S_2'$ $r \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ $S_1 \leftarrow w_1 (u_\ell' \prod_{t=1}^{k} u_{\ell,t}^{m_{\ell,t}})^r \prod_{i=1}^{t} (u_i' \prod_{t=1}^{k} u_{i,t}^{m_{i,t}})^r$ $S_2 \leftarrow w_2 \cdot g_1^r$ $\sigma = (S_1, S_2)$	$\prod_{i=1}^{\ell} A_i \stackrel{?}{=}$ $\mathbf{e}(S_1, g_2) / \mathbf{e}(S_2, \prod_{i=1}^{\ell} (\hat{u_i}' \prod_{t=1}^{k} \hat{u_{i,t}}^{m_{i,t}}))$

Figure 6.8: For setup, key generation and signature of BGLS and Sh see Figure 6.4. We denote by σ an aggregate signature on a set of ℓ messages $M_1, \ldots, M_\ell$. In LOSSW a message M_i is processed as a k-bit string denoted by $m_{i,1}, \ldots, m_{i,k}$.

6.3.3 Implementation and Performance Analysis

So far we have only considered the asymptotic performance of several batch verifiers. Unfortunately, this "paper analysis" conceals many details that are revealed only through empirical evaluation. Additionally, the existing work does not address the most important practical issue facing system implementors, namely: How a batch verifier will perform in the face of adversarial behavior such as deliberate injection of invalid signatures.

We seek to answer these questions by conducting the first empirical investigation into the feasibility of short signature batching. To conduct our experiments, we built concrete implementations of seven signature schemes described in this work, including two public key signature schemes (BLS, CHP), three

Scheme	Batch Verification Equation	Techniques
BGLS	$\mathbf{e}(\prod_{j=1}^{\eta} \sigma_j^{\delta_j}, g_2) \stackrel{?}{=} \prod_{i=1}^{\ell} \mathbf{e}(\prod_{j=1}^{\eta} H(M_{j,i})^{\delta_j}, pk_i)$	2,3
Sh	$\mathbf{e}(\prod_{j=1}^{\eta} \sigma_j^{\delta_j}, g_2) \stackrel{?}{=} \mathbf{e}(\prod_{j=1}^{\eta} H(M_{j,1}, \ldots, M_{j,\ell})^{\delta_j}, pk_{ag}) \cdot \prod_{i=1}^{\ell} \mathbf{e}(\prod_{j=1}^{\eta} H(M_{j,i}), pk_i)$	2,3
LOSSW	Let $\sigma_j = (S_{j,1}, S_{j,2})$ and $pk_i = (A_i, u_i', \hat{u_i}', u_{i,1}, \ldots, u_{i,k}, \hat{u_{i,1}}, \ldots, \hat{u_{i,k}})$ If $\eta < \ell \cdot k + 1$, $\mathbf{e}(\prod_{j=1}^{\eta} S_{j,1}^{\delta_j}, g_2) \cdot \prod_{j=1}^{\eta} \mathbf{e}(S_{j,2}^{-\delta_j}, \prod_{i=1}^{\ell} (\hat{u_i}' \prod_{t=1}^{k} \hat{u_{i,t}}^{m_{j,i,t}})) \stackrel{?}{=} \prod_{j=1}^{\eta} \prod_{i=1}^{\ell} A_i^{\delta_j}$ Otherwise, $\mathbf{e}(\prod_{j=1}^{\eta} S_{j,1}^{\delta_j}, g_2) \cdot \mathbf{e}(\prod_{j=1}^{\eta} S_{j,2}^{-\delta_j}, \prod_{i=1}^{\ell} \hat{u_i}') \cdot \prod_{i=1}^{\ell} \prod_{t=1}^{k} \mathbf{e}(\prod_{j=1}^{\eta} S_{j,2}^{-\delta_j m_{j,i,t}}, \hat{u_{i,t}}) \stackrel{?}{=} \prod_{j=1}^{\eta} \prod_{i=1}^{\ell} A_i^{\delta_j}$	2,3,4

Figure 6.9: Let η be the number of signatures to verify. The vector $(\delta_1, \ldots, \delta_\eta)$ in $\mathbb{Z}_q$ is required by the small exponents test. In LOSSW a message $M_{j,i}$ provided by pk_i in the j'th aggregate is processed as a k-bit string denoted by $m_{j,i,1}, \ldots, m_{j,i,k}$.

Identity-Based Signature schemes (ChCh, Hess, Waters), a ring signature (CYH), and a short group signature scheme (BBS). For each scheme, we measured the performance of the standard signature verification algorithm against that of the corresponding batch verifier. We then turned our attention to the problem of invalid signatures, constructing a "resilient" *divide-and-conquer* batch verifier which efficiently locates invalid signatures in a batch.

Our results lead to several surprising conclusions. First, we note that our batched Identity-Based signatures provide substantially better performance than standard (public-key) signatures, in the case where signatures are produced by different signers. This is due to a fluke of scheme construction, one that appears to stem from the related nature of IBS signing keys. Secondly, we observe that the "ideal" high-degree elliptic curve setting for short signatures (see section below) simultaneously implies both costly individual signature verification, *as well as* highly-efficient batch verifiers. Finally, we gather evidence indicating that "resilient" batch verification appears to be practical even in the presence of a substantial number of invalid signatures. The latter two results provide strong evidence for the practicality of batch verification in applications where short signatures and verification times are necessary.

Experimental Setup. To evaluate our batch verifiers, we implemented each signature scheme in C++ using the MIRACL library for elliptic curve operations [174]. Our timed experiments were conducted on a 3.0Ghz Pentium D 930 with 4GB of RAM running Linux Kernel 2.6. All hashing was implemented using SHA1,[3] and small exponents were of size 80 bits. For each scheme, our basic experiment followed the same outline: (1) generate a collection of η

[3]We selected SHA1 because the digest size closely matches the order of $\mathbb{G}_1$. It would be possible to use alternative hash functions with a similar digest size, *e.g.,* RIPEMD-160, or to truncate the output of a hash function such as SHA-256 or Whirlpool. Because the hashing time is negligible in our experiments, this should not greatly impact our results.

distinct signatures on 100-byte random message strings. (2) Conduct a timed verification of this collection using the batch verifier. (3) Repeat steps $(1, 2)$ four times, averaging to obtain a mean timing. To obtain a view of batching efficiency on collections of increasing size, we conducted the preceding test for values of η ranging from 1 to approximately 400 signatures in intervals of 20. Finally, to provide a baseline, we separately measured the performance of the corresponding *non-batched* verification, by verifying 1000 signatures and dividing to obtain the average verification time per signature. A high-level summary of our results is presented in Figure 6.11.

Curve	k	$\mathcal{R}(\mathbb{G}_1)$	$\mathcal{R}(\mathbb{G}_T)$	$\mathcal{S}_{RSA}$	Pairing Time
MNT160	6	160 bits	960 bits	960 bits	23.3 ms
MNT192	6	192 bits	1152 bits	1152 bits	33.2 ms
SS512	2	512 bits	1024 bits	957 bits	16.7 ms

Figure 6.10: Description of the elliptic curve parameters used in our experiments. $\mathcal{R}(\cdot)$ describes the approximate number of bits to optimally represent a group element. $\mathcal{S}_{RSA}$ is an estimate of "RSA-equivalent" security derived via the approach of Page et al. [155].

	Signature Size (bits)			*Individual Verification*			*Batched Verification**		
Scheme	MNT160	MNT192	SS512	MNT160	MNT192	SS512	MNT160	MNT192	SS512
Signatures									
BLS (single signer)	160	192	512	47.6 ms	77.8 ms	52.3 ms	2.28 ms	2.93 ms	32.42 ms
CHP	160	192	512	73.6 ms	119.0 ms	93.0 ms	26.16 ms	34.66 ms	34.50 ms
BLS cert + CHP sig	1280	1536	1536	121.2 ms†	196.8 ms†	145.3 ms†	28.44 ms†	37.59 ms†	66.92 ms†
Identity-Based Signatures									
ChCh	320	384	1024	49.1 ms	79.7 ms	73.3 ms	3.93 ms	5.24 ms	59.45 ms
Waters	480	576	1536	91.2 ms	138.64 ms	61.1 ms	9.44 ms	11.49 ms	59.32 ms
Hess	1120	1344	1536	49.1 ms	79.0 ms	73.1 ms	6.70 ms	8.72 ms	55.94 ms
Anonymous Signatures									
BBS (modified per §6.3.2.1)	2400	2880	3008	139.0 ms	218.3 ms	193.0 ms	24.80 ms	34.18 ms	198.03 ms
CYH, 2-member ring	480	576	1536	52.0 ms	77.0 ms	113.0 ms	6.03 ms	8.30 ms	105.69 ms
CYH, 20-member ring	3360	4032	10752	86.5 ms	126.8 ms	829.3 ms	43.93 ms	61.47 ms	932.66 ms

*Average time per verification when batching 200 signatures.
†Values were derived by manually combining data from BLS and CHP tests.

Figure 6.11: Summary of experimental results. Timing results indicate verification time *per signature*. With the exception of BLS, our experiments considered signatures generated by distinct signers. The composite scheme "BLS cert + CHP sig" describes a BLS-signed certificate on a CHP public key, along with a CHP signature.

Curve Parameters. The selection of elliptic curve parameters impacts both signature size and verification time. The two most important choices are the size of the underlying finite field $\mathbb{F}_p$, and the curve's embedding degree k. Due to the MOV attack, security is bounded by the size of the associated finite field $\mathbb{F}_{p^k}$. Simultaneously, the representation of elements $\mathbb{G}_1$ requires approximately $|p|$ bits. Thus, most of the literature on short signatures recommends choosing

a relatively small p, and a curve with a high value of k. (For example, a MNT curve with $|p| = 192$ bits and $k = 6$ is thought to offer approximately the same level of security as 1152-bit RSA [155].) The literature on short signatures focuses mainly on signature size rather than verification time, so it is easy to miss the fact that using such high-degree curves *substantially* increases the cost of a pairing operation, and thus verification time. To incorporate these effects into our results, we implemented our schemes using two high-degree ($k = 6$) MNT curves with $|p|$ equal to 160 bits and 192 bits. For completeness, we also considered a $|p|$=512 bit supersingular curve with embedding degree $k = 2$, and a subgroup $\mathbb{G}_1$ of size 2^{160}. Figure 6.10 details the curve choices along with relevant details such as pairing time and "RSA-equivalent" security determined using the approach of Page et al. [155].

6.3.3.1 Performance Results

We now present the results of our timing experiments. We first consider the two standard (public-key) signature schemes, followed by three Identity-Based alternatives. We then turn our attention to anonymous ring and group signatures. Finally, we evaluate the performance of a "resilient" batch verifier designed to verify efficiently in the presence of invalid signatures.

Public-Key signatures. Figure 6.12 presents the results of our timing experiments for the public-key BLS and CHP verifiers. Because the BLS signature does not batch efficiently for messages created by distinct signers, we considered these schemes in the combination suggested in Section 6.2.4, where BLS is used for certificates which are created by a single master authority, and CHP is used to sign the actual messages under users' individual signing keys. Surprisingly, the CHP batch verifier appears to be quite costly in the recommended MNT curve setting. This result, which is not obvious from the high-level analysis of Camenisch et al., stems from the requirement that user public keys be in the $\mathbb{G}_2$ subgroup. This necessitates expensive point operations in the curve defined over the extension field, which undoes some of the advantage gained by batching. However, even with this limitation, batching reduces the per-signature verification cost to as little as 1/3 to 1/4 that of individual verification.

Identity-Based signatures. Figure 6.13 details the results of our timing experiments for three Identity-Based signature schemes, ChCh, Waters and Hess. (For comparison, our graphs also present the non-IBS approach employing CHP signatures with BLS-signed public-key certificates.) In all experiments we consider signatures generated by different signers. We observe that in contrast with the public-key schemes, the IBSes batch quite efficiently in this case, at least when implemented in MNT curves. In particular, the Waters scheme offers surprisingly strong performance for a scheme not dependent on random oracles.[4] Note that in in our implementation of Waters, we first apply SHA1 to the message, and use the Waters hash parameter $z = 5$ which divides the

[4]However, it should be noted that Waters has a somewhat loose security reduction, and may therefore require larger parameters in order to achieve security comparable to alternative schemes.

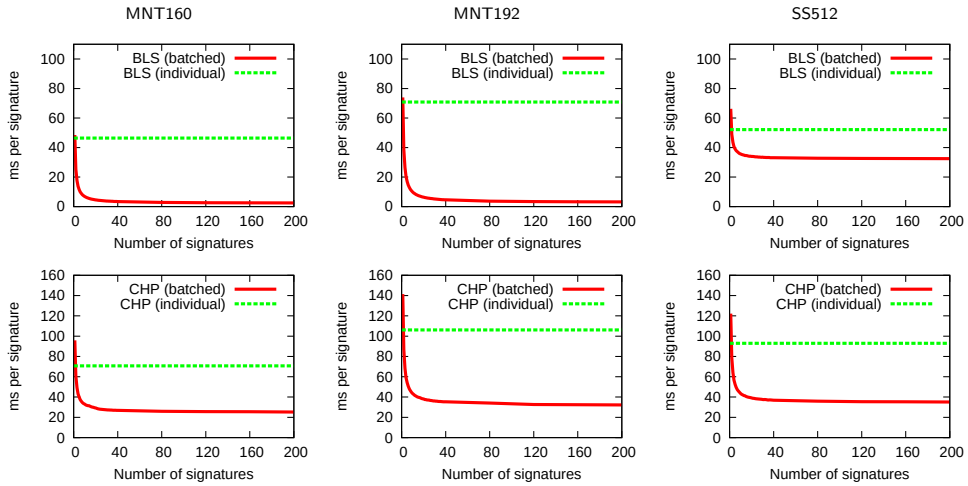


Figure 6.12: Public-Key Signature Schemes. Per-signature times were computed by dividing total batch verification time by the number of signatures verified. Note that in the BLS case, all signatures are formulated by the same signer (as for certificate generation), while for CHP each signature was produced by a different signer. Individual verification times are included for comparison.

resulting 160-bit digest into blocks of 32 bits (as proposed in [149]). Because we selected these parameters, we did not bother to implement the first case of the batch verifier, since the appropriate condition applies only for batches of size $\eta \leq 3$.

Anonymous signatures. Figure 6.14 details the results of our timing experiments for two privacy-preserving signature schemes: the CYH ring signature, and the modified BBS group signature. As is common with ring signatures, in CYH both the signature size and verification time grow linearly with the number of members in the ring. For our experiments we arbitrarily selected two cases: (1) where all signatures are formed under a 2-member ring (useful for applications such as lightweight email signing [1]), and (2) where all signatures are formed using a 20-member ring.[5] In contrast, both the signature size and verification time of the BBS group signature are independent of the size of the group. This makes group signatures like BBS significantly more practical for applications such as vehicle communication networks, where the number of signers might be quite large.

6.3.3.2 Batch Verification and Invalid Signatures

In Section 6.3.1.1, we discuss a general technique for dealing with invalid signatures encountered when batching. When batch verification fails, this *divide-and-conquer* approach recursively applies the batch verifier to individual halves

[5] Although the CYH batch verifier can easily batch signatures formed over differently-sized rings, our experiments use a constant ring size for all signatures. However our results can be considered representative of any signature collection where the *mean* ring size is 20.

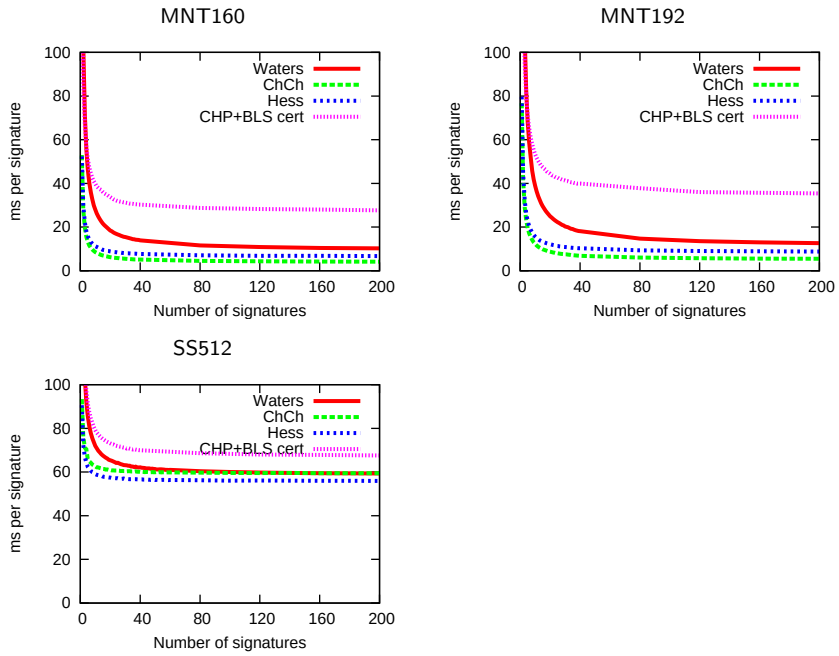


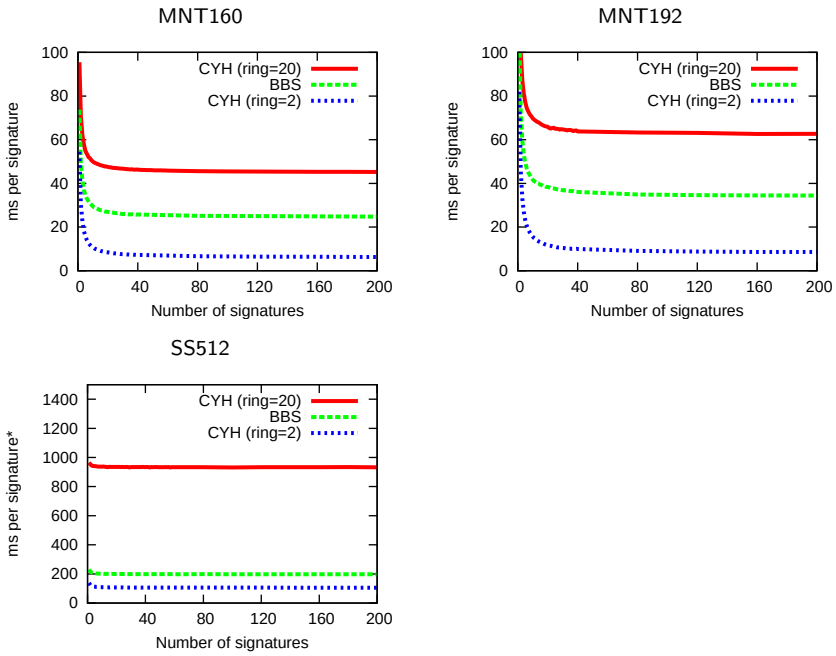
Figure 6.13: Identity-Based Signature Schemes. Times represent total batch verification time divided by the number of signatures verified. "CHP+BLS cert" represents the batched public-key alternative using certificates, and is included for comparison.

of the signature collection, until all invalid signatures have been located. To save time when recursing, we compute products of the form $\prod_{i=1}^{\eta} x_i^{\delta_i}$ so that partial products will be in place for each subset on which me might recurse. We accomplish this by placing each $x_i^{\delta_i}$ at the leaf of a binary tree and caching intermediate products at each level. This requires no additional computation, and total storage of approximately 2η group elements for each product to be computed.

To evaluate the feasibility of this technique, we used it to implement a "resilient" batch verifier for the BLS signature scheme. This verifier accepts as input a collection of signatures where some may be invalid, and outputs the index of each invalid signature found. To evaluate batching performance, we first generated a collection of 1024 valid signatures, and then randomly corrupted a r-fraction by replacing them with random group elements. We repeated this experiment for values of r ranging from 0 to 15% of the collection, collecting multiple timings at each point, and averaging to obtain a mean verification time.[6] The results of the experiment are presented in Figure 6.15.

Our results indicate that batched verification of BLS signatures is preferable to the naive individual verification algorithm even as the number of invalid signatures exceeds 10% of the total size of the batch. Note also that the random distribution of invalid signatures within the collection is nearly the worst-case

[6] Although our experiment does not re-order the signature collection, such a re-ordering need not involve memory copies and could be performed at minimal additional cost.



*Uses a different timescale.

Figure 6.14: Anonymous Signature Schemes. Times represent total batch verification time divided by the number of signatures verified. For the CYH ring signature, we consider two distinct signature collections, one consisting of 2-member rings, and another with only 20-member rings. The BBS group signature verification is independent of the group size.

for resilient verification. In many practical scenarios, invalid signatures might be grouped together within the batch (e.g., if corruption is due to a burst of EM interference). In this case, the verifier might achieve better results by omitting the random shuffle step, or by using an alternative re-ordering that is more appropriate for the setting.

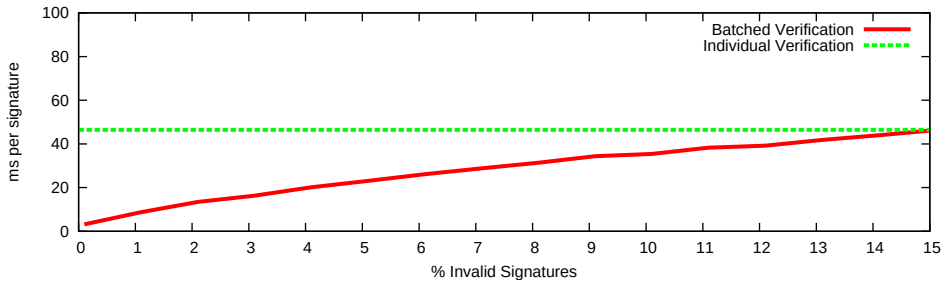


Figure 6.15: BLS batch verification in the presence of invalid signatures (160-bit MNT curve). A "resilient" BLS batch verifier was applied to a collection of 1024 purported BLS signatures, where some percentage were randomly corrupted. Per-signature times were computed by dividing the total verification time (including identification of invalid signatures) by the total number of signatures (1024), and averaging over multiple experimental runs.

Bibliography

[1] B. Adida, D. Chau, S. Hohenberger, and R. L. Rivest. Short signatures from the Weil pairing. In R. D. Prisco and M. Yung, editors, *International Conference on Security in Communication Networks – SCN 2006*, volume 4116 of *Lecture Notes in Computer Science*, pages 288–302. Springer, 2006.

[2] I. Altman. *Environment and Social Behaviour*. Brooks-Cole, 1975.

[3] J. H. An, Y. Dodis, and T. Rabin. On the security of joint signature and encryption. In L. R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 83–107. Springer, 2002.

[4] R. Anderson and M. Kuhn. Tamper resistance: A cautionary note. In *USENIX Workshop on Electronic Commerce*, pages 1–11. USENIX, 1996.

[5] R. Anderson and M. Kuhn. Low cost attacks on tamper resistant devices. In B. Christianson, B. Crispo, T. M. A. Lomas, and M. Roe, editors, *Security Protocols Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 1998.

[6] R. J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, 2001.

[7] R. J. Anderson, H. Chan, and A. Perrig. Key infection: Smart trust for smart dust. In *IEEE International Conference on Network Protocols – ICNP 2004*, pages 206–215. IEEE Computer Society, 2004.

[8] G. Ateniese, J. Camenisch, and B. de Medeiros. Untraceable RFID tags via insubvertible encryption. In *ACM Conference on Computer and Communications Security – CCS 2005*, pages 92–101. ACM, 2005.

[9] G. Ateniese, J. Camenisch, S. Hohenberger, and B. de Medeiros. Practical group signatures without random oracles, 2005. Cryptology ePrint Archive, Report 2005/385.

[10] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In M. Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270. Springer, 2000.

[11] M. H. Au, J. K. Liu, T. H. Yuen, and D. S. Wong. ID-based ring signature scheme secure in the standard model. In H. Yoshiura, K. Sakurai, K. Rannenberg, Y. Murayama, and S. Kawamura, editors, *Advances in Information and Computer Security – IWSEC 2006*, volume 4266 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2006.

[12] G. Avoine. RFID lounge - security and privacy. `http://lasecwww.epfl.ch/~gavoine/rfid/`.

[13] G. Avoine. Adversarial model for radio frequency identification, 2005. Cryptology ePrint Archive: Report 2005/049.

[14] G. Avoine, E. Dysli, and P. Oechslin. Reducing time complexity in RFID systems. In B. Preneel and S. E. Tavares, editors, *Selected Areas in Cryptography – SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 291–306. Springer, 2005.

[15] G. Avoine and P. Oechslin. A scalable and provably secure hash-based RFID protocol. In *IEEE PerCom Workshops 2005*, pages 110–114. IEEE Computer Society, 2005.

[16] D. Balfanz, D. K. Smetters, P. Stewart, and H. C. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Network and Distributed System Security Symposium – NDSS 2002*. The Internet Society, 2002.

[17] B. Barak, R. Canetti, J. B. Nielsen, and R. Pass. Universally composable protocols with relaxed set-up assumptions. In *45th Symposium on Foundations of Computer Science (FOCS)*, pages 186–195. IEEE Computer Society, 2004.

[18] J. Bardram, R. E. Kjær, and M. Ø. Pedersen. Context-aware user authentication - supporting proximity-based login in pervasive computing. In A. K. Dey, A. Schmidt, and J. F. McCarthy, editors, *Ubiquitous Computing – UbiComp 2003*, volume 2864 of *Lecture Notes in Computer Science*, pages 107–123. Springer, 2003.

[19] J. E. Bardram. The trouble with login - on usability and computer security in pervasive computing. Technical report, Center for Pervasive Computing, Aarhus, Denmark, 2003. Available from http://www.pervasive.dk/publications.

[20] N. Baric and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 480–494. Springer, 1997.

[21] K. Barr and K. Asanović. Energy aware lossless data compression. In *Mobile Systems, Applications, and Services – MobiSys 2003*, pages 231–244. USENIX, 2003.

[22] L. Batina, J. Guajardo, T. Kerins, N. Mentens, P. Tuyls, and I. Verbauwhede. An elliptic curve processor suitable for RFID-tags, 2006. Cryptology ePrint Archive, Report 2006/227.

[23] M. Bellare, J. A. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In K. Nyberg, editor, *Advances in Cryptology – EUROCRYPT 1998*, volume 1403 of *Lecture Notes in Computer Science*, pages 236–250. Springer, 1998.

[24] M. Bellare and O. Goldreich. On defining proofs of knowledge. In E. F. Brickell, editor, *Advances in Cryptology – CRYPTO 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer, 1992.

[25] M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629. Springer, 2003.

[26] M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In A. J. Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 136–153. Springer, 2005.

[27] J. C. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital signatures. In T. Helleseth, editor, *Advances in Cryptology – EUROCRYPT 1993*, volume 765 of *Lecture Notes in Computer Science*, pages 274–285. Springer, 1994.

[28] F. Bennett, T. Richardson, and A. Harter. Teleporting - making applications mobile. In *IEEE Mobile Computer Systems and Applications Workshop*, pages 82–84. IEEE Computer Society, 1994.

[29] E. Biham, O. Dunkelman, S. Indesteege, N. Keller, and B. Preneel. How to steal cars - a practical attack on keeloq. Rump Session, Crypto, 2007.

[30] R. Bird, I. S. Gopal, A. Herzberg, P. A. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of two-party authentication protocols. In J. Feigenbaum, editor, *Advances in Cryptology – CRYPTO 1991*, volume 576 of *Lecture Notes in Computer Science*, pages 44–61. Springer, 1991.

[31] D. Boneh and X. Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 2004.

[32] D. Boneh and X. Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, 2008.

[33] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In M. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2004.

[34] D. Boneh and M. K. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[35] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer, 2003.

[36] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.

[37] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.

[38] S. C. Bono, M. Green, A. Stubblefield, A. Juels, A. D. Rubin, and M. Szydlo. Security analysis of a cryptographically-enabled RFID device. In *USENIX Security Symposium – SSYM 2005*, pages 1–16. USENIX, 2005.

[39] F. Boudot. Efficient proofs that a committed number lies in an interval. In B. Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer, 2000.

[40] C. Boyd and C. Pavlovski. Attacking and repairing batch verification schemes. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 58–71. Springer, 2000.

[41] X. Boyen. Mesh signatures: How to leak a secret with unwitting and unwilling participants. In M. Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 210–227. Springer, 2007.

[42] X. Boyen and B. Waters. Compact group signatures without random oracles. In S. Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 427–444. Springer, 2006.

[43] X. Boyen and B. Waters. Full-domain subgroup hiding and constant-size group signatures. In T. Okamoto and X. Wang, editors, *Public Key Cryptography – PKC 2007*, volume 4450 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2007.

[44] M. Boyle. A shared vocabulary for privacy. UbiComp 2003: Privacy Workshop, Oct 2003.

[45] S. Brands. Untraceable off-line cash in wallets with observers. In D. R. Stinson, editor, *Advances in Cryptology – CRYPTO 1993*, volume 773 of *Lecture Notes in Computer Science*, pages 302–318. Springer, 1993.

[46] S. Brands. *Rethinking Public Key Infrastructure and Digital Certificates – Building in Privacy.* PhD Thesis, Eindhoven Inst. of Tech. The Netherlands, 1999.

[47] S. Brands and D. Chaum. Distance bounding protocols. In T. Helleseth, editor, *Advances in Cryptology – EUROCRYPT 1993*, volume 765 of *Lecture Notes in Computer Science*, pages 344–359. Springer, 1993.

[48] J. Bringer, H. Chabanne, and E. Dottax. HB^{++}: A lightweight authentication protocol secure against some attacks. In *IEEE Security, Privacy and Trust in Pervasive and Ubiquitous Computing – SecPerU 2006*, pages 28–33. IEEE Computer Society, 2006.

[49] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer. Easyliving: Technologies for intelligent environments. In P. Thomas and H. W. Gellersen, editors, *Handheld and Ubiquitous Computing – HUC 2000*, volume 1927 of *Lecture Notes in Computer Science*, pages 12–29. Springer, 2000.

[50] M. Burmester, B. de Medeiros, and R. Motta. Robust, anonymous RFID authentication with constant key-lookup, 2007. Cryptology ePrint Archive: Report 2007/402.

[51] M. Burmester, T. van Le, and B. de Medeiros. Provably secure ubiquitous systems: Universally composable RFID authentication protocols, 2006. Cryptology ePrint Archive: Report 2006/131.

[52] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems – TOCS 1990*, 8(1):18–36, 1990.

[53] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich. How to win the clonewars: Efficient periodic n-times anonymous authentication. In *ACM Conference on Computer and Communications Security – CCS 2006*, pages 201–210. ACM, 2006.

[54] J. Camenisch, S. Hohenberger, and M. Ø. Pedersen. Batch verification of short signatures. In M. Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 246–263. Springer, 2007.

[55] J. Camenisch, S. Hohenberger, and M. Ø. Pedersen. Batch verification of short signatures, 2007. Cryptology ePrint Archive: Report 2007/172.

[56] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer, 2001.

[57] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 61–76. Springer, 2002.

[58] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In M. K. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer, 2004.

[59] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In B. S. K. Jr., editor, *Advances in Cryptology – CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 1997.

[60] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *ACM Symposium on Theory of Computing – STOC 1998*, pages 209–218. ACM, 1998.

[61] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 255–271. Springer, 2003.

[62] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 404–421. Springer, 2005.

[63] T. Cao, D. Lin, and R. Xue. Security analysis of some batch verifying signatures from pairings. *International Journal of Network Security*, 3(2):138–143, 2006.

[64] Car2Car Communication Consortium. `http://car-to-car.org`.

[65] J. C. Cha and J. H. Cheon. An identity-based signature from gap Diffie-Hellman groups. In Y. Desmedt, editor, *Public Key Cryptography – PKC 2003*, Lecture Notes in Computer Science, pages 18–30. Springer, 2003.

[66] S. Chatterjee and P. Sarkar. Trading time for space: Towards an efficient IBE scheme with short(er) public parameters in the standard model. In D. Won and S. Kim, editors, *Information Security and Cryptology – ICISC 2005*, volume 3935 of *Lecture Notes in Computer Science*, pages 424–440. Springer, 2005.

[67] S. Chatterjee and P. Sarkar. HIBE with short public parameters without random oracle. In X. Lai, editor, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 145–160. Springer, 2006.

[68] D. Chaum. Security without identification: transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[69] D. Chaum and J.-H. Evertse. A secure and privacy-protecting protocol for transmitting personal information between organizations. In A. M. Odlyzko, editor, *Advances in Cryptology – CRYPTO 1986*, volume 263 of *Lecture Notes in Computer Science*, pages 118–167. Springer, 1986.

[70] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In S. Goldwasser, editor, *Advances in Cryptology – CRYPTO 1988*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer, 1990.

[71] D. Chaum and E. van Heyst. Group signatures. In S. Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer, 1991.

[72] L. Chen, Z. Cheng, and N. P. Smart. Identity-based key agreement protocols from pairings. *Journal of Information Security*, 6(4):213–241, 2007.

[73] J. H. Cheon, Y. Kim, and H. J. Yoon. A new ID-based signature with batch verification, 2004. Cryptology ePrint Archive: Report 2004/131.

[74] H. Cheung. How to: Building a bluesniper rifle. `http://www.tomsguide.com/us/how-to-bluesniper-pt1,review-408.html`, 2005.

[75] S. S. M. Chow, S.-M. Yiu, and L. C. Hui. Efficient identity based ring signature. In J. Ioannidis, A. D. Keromytis, and M. Yung, editors, *Applied Cryptography and Network Security – ACNS 2005*, volume 3531 of *Lecture Notes in Computer Science*, pages 499–512. Springer, 2005.

[76] H. B. Christensen and J. Bardram. Supporting human activities - exploring activity-centered computing. In G. Borriello and L. E. Holmquist, editors, *Ubiquitous Computing – UbiComp 2002*, volume 2498 of *Lecture Notes in Computer Science*, pages 107–116. Springer, 2002.

[77] M. D. Corner and B. D. Noble. Zero-interaction authentication. In *Mobile Computing and Networking – MobiCom 2002*, pages 1–11. ACM, 2002.

[78] R. Cramer and I. Damgård. Fast and secure immunization against adaptive man-in-the-middle impersonation. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 75–87. Springer, 1997.

[79] I. Damgård. Payment systems and credential mechanism with provable security against abuse by individuals. In S. Goldwasser, editor, *Advances*

in Cryptology – CRYPTO 1988, volume 403 of *Lecture Notes in Computer Science*, pages 328–335. Springer, 1990.

[80] I. Damgård, K. Dupont, and M. Ø. Pedersen. Unclonable group identification. In S. Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 555–572. Springer, 2006.

[81] I. Damgård and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Y. Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 125–142. Springer, 2002.

[82] I. Damgård and M. Ø. Pedersen. RFID security: Tradeoffs between security and efficiency. In T. Malkin, editor, *Topics in Cryptology – CT-RSA 2008*, volume 4964 of *Lecture Notes in Computer Science*, pages 318–332. Springer, 2008.

[83] D. E. Denning and P. D. MacDoran. Location-based authentication: Grounding cyberspace for better security. *Computer Fraud and Security*, Feb 1996.

[84] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[85] P. Dourish, R. E. Grinter, J. D. de la Flor, and M. Joseph. Security in the wild: User strategies for managing security as an everyday, practical problem. *Personal and Ubiquitous Computing*, 8(6):391–401, 2004.

[86] P. Droz, C. GÃ¼lcÃ¼, and R. Haas. WANTED: A theft-deterrent solution for the pervasive computing world. In *IEEE International Conference on Computer Communications and Networks – ICCCN 2000*, pages 374–379. IEEE Computer Society, 2000.

[87] Ducatel, Bogdanowicz, Scapolo, Leijten, and Burgelman. Scenarios for ambient intelligence in 2010, Feb 2001.

[88] Ensure technologies. `http://www.ensuretech.com`.

[89] evgeniy Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In S. Vaudenay, editor, *Public Key Cryptography – PKC 2005*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431. Springer, 2005.

[90] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.

[91] A. L. Ferrara, M. Green, S. Hohenberger, and M. Ø. Pedersen. Practical short signature batch verification, 2008. Cryptology ePrint Archive: Report 2008/015.

[92] A. Fiat. Batch RSA. In G. Brassard, editor, *Advances in Cryptology – CRYPTO 1989*, volume 435 of *Lecture Notes in Computer Science*, pages 175–185. Springer, 1989.

[93] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology – CRYPTO 1986*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[94] M. Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 152–168. Springer, 2005.

[95] M. P. Fossorier, M. J. Mihaljević, H. Imai, Y. Cui, , and K. Matsuura. A novel algorithm for solving the LPN problem and its application to security evaluation of the HB protocol for RFID authentication, 2006. Cryptology ePrint Archive: Report 2006/197.

[96] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers, 2006. Cryptology ePrint Archive: Report 2006/165.

[97] R. Gavison. Privacy and the limits of the law. In *Computers, Ethics, and Social Values*, pages 332–351. Prentice Hall, 1995.

[98] R. Gennaro and P. Rohatgi. How to sign digital streams. In B. S. K. Jr., editor, *Advances in Cryptology – CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 180–197. Springer, 1997.

[99] C. Gentry. How to compress rabin ciphertexts and signatures (and more). In M. K. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 179–200. Springer, 2004.

[100] C. Gentry and Z. Ramzan. Identity-based aggregate signatures. In M. Yung, editor, *Public Key Cryptography – PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 257–273. Springer, 2006.

[101] H. Gilbert, M. Robshaw, and H. Sibert. An active attack against HB+ - a provably secure lightweight authentication protocol, 2007. Cryptology ePrint Archive: Report 2005/237.

[102] H. Gilbert, M. J. Robshaw, and Y. Seurin. HB#: Increasing the security and efficiency of hb+, 2008. Cryptology ePrint Archive: Report 2008/028.

[103] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. In *25th Symposium on Foundations of Computer Science (FOCS)*, pages 464–479. IEEE Computer Society, 1984.

[104] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of ACM*, 38(3):691–729, 1991.

[105] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *ACM Symposium on Theory of Computing – STOC 1985*, pages 291–304. ACM, 1985.

[106] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[107] R. Granger and N. P. Smart. On computing products of pairings, 2006. Cryptology ePrint Archive: Report 2006/172.

[108] G. P. Hancke and M. G. Kuhn. An RFID distance bounding protocol. In *IEEE Security and Privacy for Emerging Areas in Communications Networks – SECURECOMM 2005*, pages 67–73. IEEE Computer Society, 2005.

[109] L. Harn. Batch verifying multiple DSA digital signatures. *Electronics Letters*, 34(9):870–871, 1998.

[110] L. Harn. Batch verifying multiple RSA digital signatures. *Electronics Letters*, 34(12):1219–1220, 1998.

[111] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The anatomy of a context-aware application. In *Mobile Computing and Networking – MobiCom 1999*, pages 59–68. ACM, 1999.

[112] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The anatomy of a context-aware application. In *ACM/IEEE Conference on Mobile Computing and Networking – MOBICOM 1999*, pages 59–68. ACM, 1999.

[113] M. E. Hellman. A cryptanalytic time-memory tradeoff. *IEEE Transactions on Information Theory*, 26(6):401–406, 1980.

[114] F. Hess. Efficient identity based signature schemes based on pairings. In K. Nyberg and H. M. Heys, editors, *Selected Areas in Cryptography – SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 310–324. Springer, 2002.

[115] N. J. Hopper and M. Blum. Secure human identification protocols. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2001.

[116] F. Hoshino, M. Abe, and T. Kobayashi. Lenient/strict batch verification in several groups. In G. I. Davida and Y. Frankel, editors, *Information Security*, Lecture Notes in Computer Science, pages 81–94. Springer, 2001.

[117] M.-S. Hwang, C.-C. Lee, and Y.-L. Tang. Two simple batch verifying multiple digital signatures. In S. Qing, T. Okamoto, and J. Zhou, editors, *Information and Communications Security – ICICS 2001*, Lecture Notes in Computer Science, pages 233–237. Springer, 2001.

[118] M.-S. Hwang, I.-C. Lin, and K.-F. Hwang. Cryptanalysis of the batch verifying multiple RSA digital signatures. *Informatica, Lithuanian Academy of Sciences*, 11(1):15–19, 2000.

[119] IBM JCOP. http://www.zurich.ibm.com/csc/infosec/smartcard.html.

[120] IEEE. 5.9 GHz dedicated short range communications. `http://grouper.ieee.org/groups/scc32/dsrc`.

[121] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In U. Maurer, editor, *Advances in Cryptology – EUROCRYPT 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 143–154. Springer, 1996.

[122] A. Juels. RFID security and privacy: a research survey. *IEEE Journal on Selected Areas in Communications*, 24(2):381–394, 2006.

[123] A. Juels, R. L. Rivest, and M. Szydlo. The blocker tag: Selective blocking of RFID tags for consumer privacy. In *ACM Conference on Computer and Communications Security – CCS 2003*, pages 103–111. ACM, 2003.

[124] A. Juels and S. Weis. Authenticating pervasive devices with human protocols. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3126 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005.

[125] A. Juels and S. A. Weis. Defining strong privacy for RFID. In *IEEE PerCom Workshops 2007*, pages 342–347. IEEE Computer Society, 2007.

[126] B. Kaliski. PKCS #5: Password-based cryptography specification version 2.0. RFC 2898 (Proposed Standard), Sep 2000.

[127] M. Kanellos. New Wi-Fi distance record: 382 kilometers. `http://www.news.com/8301-10784_3-9730708-7.html`, 2007.

[128] A. Kiayias and M. Yung. Group signatures with efficient concurrent join. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 198–214. Springer, 2005.

[129] J. Kilian and E. Petrank. Identity escrow. In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO 1998*, volume 1462 of *Lecture Notes in Computer Science*, pages 169–187. Springer, 1998.

[130] N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. In N. P. Smart, editor, *IMA International Conference – Cryptography and Coding 2005*, volume 3796 of *Lecture Notes in Computer Science*, pages 13–36. Springer, 2005.

[131] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. RFC 2104 (Proposed Standard), Feb 1997.

[132] C.-S. Laih and S.-M. Yen. Improved digital signature suitable for batch verification. *IEEE Transactions on Computers*, 44(7):957–959, 1995.

[133] O. Landsiedel, K. Wehrle, and S. Götz. Accurate prediction of power consumption in sensor networks. In *IEEE Embedded Networked Sensors – EmNets 2005*, pages 37–44. IEEE Computer Society, 2005.

[134] L. Law and B. J. Matt. Finding invalid signatures in pairing-based batches. In S. D. Galbraith, editor, *IMA International Conference – Cryptography and Coding 2007*, volume 4887 of *Lecture Notes in Computer Science*, pages 34–53. Springer, 2007.

[135] S. Lederer, J. Mankoff, and A. K. Dey. Towards a deconstruction of the privacy space. UbiComp 2003: Privacy Workshop, Oct 2003.

[136] S. Lee, S. Cho, J. Choi, and Y. Cho. Efficient identification of bad signatures in RSA-type batch signature. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 89-A(1):74–80, 2006.

[137] C. Lim and P. Lee. Security of interactive DSA batch verification. In *Electronics Letters*, volume 30(19), pages 1592–1593, 1994.

[138] C. H. Lim. Efficient multi-exponentation and application to batch verification of digital signatures, 2000. `http://dasan.sejong.ac.kr/~chlim/english_pub.html`.

[139] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. Sequential aggregate signatures and multisignatures without random oracles. In S. Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 465–485. Springer, 2006.

[140] A. Lysyanskaya. *Signature Schemes and Applications to Cryptographic Protocol Design*. PhD Thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 2002.

[141] A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In C. Adams and H. Heys, editors, *Selected Areas in Cryptography – SAC 1999*, volume 1758 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 1999.

[142] A. Lysyanskaya, R. Tamassia, and N. Triandopoulos. Multicast authentication in fully adversarial networks. In *IEEE Symposium on Security and Privacy – S&P 2004*. IEEE Computer Society, 2004.

[143] A. Menezes, S. Vanstone, and T. Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *ACM Symposium on Theory of Computing – STOC 1991*, pages 80–89. ACM, 1991.

[144] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *Foundations of Computer Science – FOCS 1999*, pages 120–130. IEEE Computer Society, 1999.

[145] Ministeriet for Videnskab, Teknologi og Udvikling. Teknologisk Fremsyn – Pervasive Computing, Jan 2003.

[146] D. Molnar, A. Soppera, and D. Wagner. A scalable, delegatable pseudonym protocol enabling ownership transfer of RFID tags. In B. Preneel and S. E. Tavares, editors, *Selected Areas in Cryptography – SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 276–290. Springer, 2005.

[147] J. Monnerat and S. Vaudenay. Undeniable signatures based on characters: How to sign with one bit. In F. Bao, R. H. Deng, and J. Zhou, editors, *Public Key Cryptography – PKC 2004*, volume 2947 of *Lecture Notes in Computer Science*, pages 69–85. Springer, 2004.

[148] J. Monnerat and S. Vaudenay. Short 2-move undeniable signatures. In P. Q. Nguyen, editor, *Progress in Cryptology – VIETCRYPT 2006*, volume 4341 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 2006.

[149] D. Naccache. Secure and practical identity-based encryption, 2005. Cryptology ePrint Archive: Report 2005/369.

[150] D. Naccache, D. M'Raïhi, S. Vaudenay, and D. Raphaeli. Can DSA be improved? Complexity trade-offs with the digital signature standard. In A. D. Santis, editor, *Advances in Cryptology – EUROCRYPT 1994*, volume 950 of *Lecture Notes in Computer Science*, pages 77–85. Springer, 1994.

[151] K. Nagel, C. D. Kidd, T. O'Connell, A. Dey, and G. D. Abowd. The family intercom: Developing a context-aware audio communication system. In G. D. Abowd, B. Brumitt, and S. Shafer, editors, *Ubiquitous Computing – UbiComp 2001*, volume 2201 of *Lecture Notes in Computer Science*, pages 176–183. Springer, 2001.

[152] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *Foundations of Computer Science – FOCS 1997*, pages 458–467. IEEE Computer Society, 1997.

[153] Y. Nohara, S. Inoue, K. Baba, and H. Yasuura. Quantitative evaluation of unlinkable id matching schemes. In *ACM Workshop on Privacy in the Electronic Society – WPES 2005*, pages 55–60. ACM, 2005.

[154] M. Ohkubo, K. Suzuki, and S. Kinoshita. Efficient hash-chain based RFID privacy protection scheme. UbiComp 2004: Privacy Workshop, Sep 2004.

[155] D. Page, N. Smart, and F. Vercauteren. A comparison of MNT curves and supersingular curves. *Applicable Algebra in Engineering, Communication and Computing*, 17(5):379–392, 2006.

[156] J. I. Pagter and M. Ø. Pedersen. The all-or-nothing anti-theft policy - theft protection for pervasive computing. In *IEEE Advanced Information Networking and Applications – AINA 2007*, pages 626–631. IEEE Computer Society, 2007.

[157] J. I. Pagter and M. G. Petersen. A sense of security in pervasive computing - is the light on when the refrigerator door is closed? In S. Dietrich and R. Dhamija, editors, *Financial Cryptography 2007*, volume 4886 of *Lecture Notes in Computer Science*, pages 383–388. Springer, 2007.

[158] L. Palen and P. Dourish. Unpacking "privacy" for a networked world. In *Human factors in computing systems – SIGCHI 2003*, pages 129–136. ACM, 2003.

[159] J. Pastuszak, D. Michatek, J. Pieprzyk, and J. Seberry. Identification of bad signatures in batches. In H. Imai and Y. Zheng, editors, *Public Key Cryptography – PKC 2000*, volume 1751 of *Lecture Notes in Computer Science*, pages 28–45. Springer, 2000.

[160] M. Ø. Pedersen, J. I. Pagter, and T. P. Pedersen. Pervasive computing: IT security and privacy, Nov 2004.

[161] A. Perrig, R. Canetti, D. X. Song, and J. D. Tygar. Efficient and secure source authentication for multicast. In *Network and Distributed System Security Symposium – NDSS 2001*. The Internet Society, 2001.

[162] O. Perron. Bemerkungen über die verteilung der quadratischen reste. *Mathematische Zeitschrift*, 56(2):122–130, 1952.

[163] Pointsec. Taxis hailed as black hole for lost cell phones and PDAs, as confidential data gets taken for a ride. `http://www.checkpoint.com/press/pointsec/2005/01-24a.html`, 2006.

[164] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1979.

[165] M. Raya and J.-P. Hubaux. Securing vehicular ad hoc networks. *Journal of Computer Security*, 15(1):39–68, 2007.

[166] RFID Guardian. `http://www.rfidguardian.org/`.

[167] R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565. Springer, 2001.

[168] D. M. Russell, C. Drews, and A. Sue. Social aspects of using large public interactive displays for collaboration. In G. Borriello and L. E. Holmquist, editors, *Ubiquitous Computing – UbiComp 2002*, volume 2498 of *Lecture Notes in Computer Science*, pages 229–236. Springer, 2002.

[169] D. M. Russell and R. Gossweiler. On the design of personal & communal large information scale appliances. In G. D. Abowd, B. Brumitt, and S. Shafer, editors, *Ubiquitous Computing – UbiComp 2001*, volume 2201 of *Lecture Notes in Computer Science*, pages 354–361. Springer, 2001.

[170] B. RÃ¶ssler. *The Value of Privacy*. Polity Press, 2005.

[171] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. In *Symposium on Cryptography and Information Security – SCIS 2000*, 2000.

[172] B. Schneier. *Secrets and Lies: Digital Security in a Networked World*. John Wiley & Sons, 2000.

[173] C.-P. Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, *Advances in Cryptology – CRYPTO 1989*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1989.

[174] M. Scott. Multiprecision integer and rational arithmetic C/C++ library (MIRACL). Published by Shamus Software Ltd., `http://www.shamus.ie/`.

[175] SeVeCom. Security on the road. `http://www.sevecom.org`.

[176] Z. Shao. Enhanced aggregate signatures from pairings. In M. Yung, editor, *Information Security and Cryptology – CISC 2005*, volume 3822 of *Lecture Notes in Computer Science*, pages 140–149. Springer, 2005.

[177] D. K. Smetters and R. E. Grinter. Moving from the design of usable security technologies to the design of useful secure applications. In *New Security Paradigms Workshop - NSPW 2002*, pages 82–89. ACM, 2002.

[178] M. T. Smith. Smart cards: Integrating for portable complexity. *IEEE Computer*, 31(8):110–115, 1998.

[179] F. Stajano and R. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In B. Christianson, B. Crispo, J. A. Malcolm, and M. Roe, editors, *Security Protocols Workshop*, volume 1796 of *Lecture Notes in Computer Science*, pages 172–194. Springer, 1999.

[180] M. Stanek. Attacking LCCC batch verification of RSA signatures, 2006. Cryptology ePrint Archive: Report 2006/111.

[181] M. Steil. 17 mistakes microsoft made in the xbox security system. `http://www.xbox-linux.org/wiki/17_Mistakes_Microsoft_Made_in_the_Xbox_Security_System`, 2005.

[182] B. Stone. Pinch my ride. `http://www.wired.com/wired/archive/14.08/carkey.html`, 2006.

[183] M. Tompa and H. Woll. Random self-reducibility and zero knowledge interactive proofs of possession of information. In *Foundations of Computer Science – FOCS 1987*, pages 472–482. IEEE Computer Society, 1987.

[184] J. Trevor, D. M. Hilbert, and B. N. Schilit. Issues in personalizing shared ubiquitous devices. In G. Borriello and L. E. Holmquist, editors, *Ubiquitous Computing – UbiComp 2002*, volume 2498 of *Lecture Notes in Computer Science*, pages 56–72. Springer, 2002.

[185] G. Tsudik. YA-TRAP: Yet another trivial RFID authentication protocol. In *IEEE PerCom Workshops 2006*, pages 640–643. IEEE Computer Society, 2006.

[186] J. Viega and D. McGrew. The use of galois/counter mode (GCM) in IPsec encapsulating security payload (ESP). RFC 4106 (Proposed Standard), Jun 2005.

[187] R. Want, A. Hopper, V. FalcÃ£o, and J. Gibbons. The active badge location system. *ACM Transactions on Information Systems – TOIS 1992*, 10(1):91–102, 1992.

[188] A. Ward, A. Jones, and A. Hopper. A new location technique for the active office. *IEEE Personal Communication Magazine*, 4(5):42–47, 1997.

[189] B. Waters. Efficient identity-based encryption without random oracles. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 320–329. Springer, 2005.

[190] S. A. Weis, S. E. Sarma, R. L. Rivest, and D. W. Engels. ecurity and privacy aspects of low-cost radio frequency identification systems. In B. Preneel and S. E. Tavares, editors, *Security in Pervasive Computing – SPC 2003*, volume 2802 of *Lecture Notes in Computer Science*, pages 201–212. Springer, 2003.

[191] A. Whitten and J. Tygar. Why johnny can't encrypt: A usability evaluation of PGP 5.0. In *USENIX Security Symposium*, pages 169–184. USENIX, 1999.

[192] H. Yoon, J. H. Cheon, and Y. Kim. Batch verifications with ID-based signatures. In C. Park and S. Chee, editors, *Information Security and Cryptology – ICISC 2004*, Lecture Notes in Computer Science, pages 233–248. Springer, 2004.

[193] F. Zhang and K. Kim. Efficient ID-based blind signature and proxy signature from bilinear pairings. In R. Safavi-Naini and J. Seberry, editors, *Information Security and Privacy, Australasian Conference – ACISP 2003*, volume 2727 of *Lecture Notes in Computer Science*, pages 312–323. Springer, 2003.

[194] F. Zhang, R. Safavi-Naini, and W. Susilo. Efficient verifiably encrypted signature and partially blind signature from bilinear pairings. In T. Johansson and S. Maitra, editors, *Progress in Cryptology – INDOCRYPT 2003*, volume 2904 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 2003.