Open access Journal International Journal of Emerging Trends in Science and Technology

Network Security Cryptographic Protocols and Lattice Problems

Author

Dr Daruri Venugopal

M.Sc;B.Ed; M.Sc;M.Phill;M.Tech;Ph.D.(Post.Doct.);LMISTE,PGDCJ,

Dept.of Computer Science & Engineering, Siddhartha Institute of Technology And Sciences

Narapally, Ghatkesar, R.R.Dist.

Email: *Profdarurivg.edu@gmail.com*

Abstract

In this thesis we present new results in two areas – cryptographic protocols and lattice problems.

- *We present a new protocol for electronic cash which is designed to function on hardware with limited computing power. The scheme has provable security properties and low computational requirements, but it still gives a fair amount of privacy. Another feature of the system is that there is no master secret that could be used for counterfeiting money if stolen.*
- *We introduce the notion of hierarchical group signatures. This is a proper generalization of group signatures, which allows multiple group managers organized in a tree with the signers as leaves. For a signer that is a leaf of the sub tree of a group manager, the group manager learns which of its children that (perhaps indirectly) manages the signer. We provide definitions for the new notion and construct a scheme that is provably secure given the existence of a family of trapdoor permutations. We also present a construction which is relatively practical, and prove its security in the random oracle model under the strong RSA assumption and the DDH assumption.*
- *We show a weakness in the specification for offline capable EMV payment cards. The weakness, which applies to cards without RSA capability, enables an attacker to duplicate a card and make transactions that cannot be tied to the original card.*
- *We give a method for approximating any n-dimensional lattice with a lattice Λ whose factor group Z^n/Λ has $(n-1)$ cycles of equal length with arbitrary precision. We also show that a direct consequence of this is that the Shortest Vector Problem and the Closest Vector Problem cannot be easier for this type of lattices than for general lattices.*

Keywords: *Security properties; Counterfeiting; Hierarchical group ; RSA assumption ; DDH assumption; EMV payment cards; Dimensional lattice ; Arbitrary precision*

1 Introduction

1.1 Confidentiality and Authenticity

When the word "cryptography" is mentioned, what first comes to mind is probably sending secret messages. This is justified, as hiding information from eavesdroppers, confidentiality, is the traditional reason to use cryptography. An analogy is to send a message in a sealed envelope (or maybe in a locked safe, although it is debatable how realistic such an analogy is). Sometimes we are not primarily interested in hiding information, but rather in ensuring that information isn't modified or counterfeited, authenticity. By this we mean that the receiver can be convinced that sender is who he claims to be, and that the message has not been altered on the

way. The analogy here is to sign a paper with the message on it. Since signatures are assumed to be hard to forge, a signature identifies the sender.

In an environment where messages are mainly sent electronically, we need meth-ods to achieve confidentiality and authenticity by digital means, and this is one major part of what cryptographic research is about. The traditional approach is to set up a key k and define a function E to encrypt and a function D to decrypt so that $D_k(E_k(m)) = m$ for any legal message m. We will call m the plaintext and the encryption $E_k(m)$ the ciphertext.

Since we want the system to be secure, we want it to be infeasible to compute any useful information about the plaintext from the ciphertext, provided that the key k is unknown.

We even want it to be infeasible if certain side information is known, such as a subset of legal messages from which m is drawn, or encryptions of other messages under the same key. Consider the functions necessary to ensure that a message isn't counterfeited or modified. The usual approach is to define a function S to create a message authentication code (MAC) and a function V to verify that a MAC is valid. The function S takes as input a message m and a key k and returns a MAC. The function V takes a key, a message and a MAC, and returns 1 if the MAC is valid and 0 otherwise. It must hold that $V_k(m, S_k(m)) = 1$, and it should be infeasible to compute a message m and a MAC s such that $V_k(m, s)$ without knowledge of k. Also here the attacker may have access to side information such as MACs on messages of his choice.

1.2 Public Key Cryptography Asymmetric Encryption Schemes

In the above definition, the same key is used for encryption and decryption. For a long time, this was the only known way to perform cryptography. In the middle of the 1970s, a major breakthrough was made when methods to perform asymmetric cryptography were discovered. Asymmetric systems use two keys, the public key, pk and the private key (sometimes called **secret key**), sk . The public key is used to encrypt, and the private key to decrypt so that $D_{sk}(E_{pk}(m)) = m$. The public key can be published, since it is used only for encryption, but the private key must be kept secret.

Let us now compare this with symmetric cryptosystems to see what the differences may mean in practice. Assume ten people work at the same company, and that they want to be able to send encrypted messages to each other. First con-sider a symmetric cryptosystem.

One solution is to have a single common key that everything is encrypted with, but there are several drawbacks with this approach. Someone who gets hold of the key (for example by bribing one of the employees) is able to read all messages sent. Also any employee can read any message, even it wasn't meant for him. If an employee quits, a new key has to be set up and distributed in a secure manner. A second solution is to have one key between every pair of employees. Then only the intended recipient can read his messages, and if one employees sells (or accidently discloses) his keys, only the messages sent or received by that employee can be read. However, the number of keys necessary for such a system is high. Our ten employees need a total of 45 keys. Although this number may not seem very high, we must take into account that agreeing on a symmetric key is a cumbersome task.

It is not advisable to the keys electronic-ally, since they can be eavesdropped, and if a key is sent by mail, there is always the risk that someone opens the envelope and gets the key. The only safe way is to meet in person. Now consider a company with 1000 employees. Then a total of 499, 500 keys are necessary! It is obvious that symmetric cryptosystems have certain drawbacks.

Now let us consider using asymmetric cryptography to solve the problem. Each of the ten employees generates a key pair consisting of a private and a public key. The public keys are published, say in the company phone book. If

Alice wants to send a message to Bob, she looks up Bob in the phone book, encrypts using his public key and sends the message. Bob uses his private key to decrypt, and no-one else can read the message. If the company hires new employees, each of them generates a key pair. No keys have to be exchanged under secure conditions.

1.3. Building Ccryptographic Protocols

Digital Signatures

Also authenticity can be achieved by asymmetric means. When a MAC is used, the same key is used for computing the MAC and and verifying it. Therefore only the intended recipient can check the validity of the message. Furthermore, ability to verify implies ability to compute a MAC, making it hard to use a signature as proof in case of a dispute. Therefore, in many situations, it is desirable to have a scheme in which it is possible to verify without being able to sign. Using asymmetric techniques we can construct a scheme where the signing is performed using the private key sk and the verification with the public key pk . Now it must hold that V_{pk} (m, S_{sk} (m)) = 1.

This is also what we expect from real-world signing schemes – anyone can look at a signature and check whether it has been written by the putative sender (by comparing it with other signatures written by the same person), but no-one but the sender else should be able to produce such a signature.

A digital signature is in one sense more secure than a physical signature on paper. When a paper with the message written on it is signed, it is hard to ensure that the message is not altered afterwards. A forger may add new text to a signed document or combine pages from two or more signed documents into a new document. A secure digital signature scheme withstands attacks of this type, since the signature is tied to the message and becomes invalid if the message is modified. Two of the most important building blocks for cryptographic functions are one-way functions, i.e., functions that are easy to compute but hard to invert, trapdoor functions,

i.e., functions that are one-way functions with the additional property that there is a secret which makes the function easy to invert. Take, for example, multiplication. It is easy to multiply two numbers, but no method is known that factors a numbers into its prime factors in reasonable time. It should be noted that the existence of one-way and trapdoor functions is a classical open problem, and a proof of their existence would be a major breakthrough. However, there are functions that have been subject to intensive research for more than thirty years, and no evidence contradicting the hypothesis that they are trapdoor functions have been found.

It is therefore reasonable to assume that they are indeed trapdoor functions. From functions that are assumed to be trapdoor functions, it is possible to build cryptographic primitives, e.g., encryption and signature schemes. To achieve more complex tasks, such as setting up a secure channel between parties who have not previously met or creating digital coins, we need to describe how to combine primitives to get the functionality we need. The result is called a protocol, and the protocol describes how the participants should act. A protocol can be seen as a set of algorithms, one for each participant.

A protocol may be interactive or non-interactive. An interactive protocol is used when the parties can send messages to each other in an interleaved manner. An example may be a user logging on to a web-site. In a non-interactive protocol the sender creates the message on his own, and only then sends it to the receiver. Encrypting and signing emails are a typical examples of non-interactive protocols.

1.4 Efficient vs. Practical Protocols

Naturally we want our protocols to be as efficient as possible. However, in different contexts efficiency may have different meanings. The common definition of an efficient algorithm is that

the execution time is bounded by a polynomial in the size of the input. For example, the grade school algorithm for multiplication is polynomial time, since the number of steps needed is less than $2n^2$, where n is the number of digits of each factor.

An example of an algorithm that is not polynomial is factoring by exhaustive search. To factor an n-bit number m we may need to check each number up to $\sqrt{m}$, that is, $2^{n/2}$ different numbers. Even if we assume that we can check divisibility in a single step, we still need an exponential number of steps before we are guaranteed to have a result.

It is clear that this definition of efficient algorithms does not cover everything we need from an algorithm to be usable in practice. If we design an algorithm that runs in n^{30} steps, it would still be considered efficient according to the above definition. However, the algorithm would be impossible to use in practice except for extremely small inputs.

In this thesis we focus on protocols that are not only efficient in the above meaning, but that are practical. Therefore the protocols must be specified in such detail that it is possible to analyze their running time precisely and not only show that it is bounded by some polynomial. Also, being practical is not a strict definition. In some cases, we want a protocol that can be executed on devices with little computing power such as smart-cards or mobile phones.

In other cases it is enough if the protocol runs reasonably fast on a personal computer, and in still other cases the protocol will run on a server with large storage capabilities.

1.5 Security of Cryptographic Primitives and Protocols

Obviously we want the cryptographic primitives we use to be secure. However, we need to define precisely what we mean by security of a primitive.

Let us consider an encryption scheme. One definition of security is that the scheme is secure if an attacker who sees a ciphertext cannot recover the plaintext. However, in some scenarios this is not enough, since the attacker may have access to additional in-formation. Maybe the attacker knows that the plaintext is either “yes” or “no”, and maybe the attacker has seen encryptions of other plaintexts. Maybe the attacker even has seen encryptions of “yes” and “no”.

A good cryptosystem should remain secure even under these circumstances. For example, to remain secure even if the attacker knows encryptions of “yes” and “no”, the encryption must be probabilistic.

1.6. ANONYMITY

Designing protocols that are as secure as the primitives used is not trivial. It may very well be the case that a protocol turns out to be insecure although all components used are secure. Also in the case of protocols, the term “secure” must be properly defined. Take, for example, a scheme for electronic cash involving customers, merchants and a bank. Naturally a customer should not be able to counterfeit money, but what happens if a customer and a merchant collaborates to produce counterfeit money? Or maybe when two customers together try to create a coin that appears to be valid to the merchant but which is rejected by the bank? Obviously there are many subtle details when deciding what kind of security we want from a protocol. Therefore it is important to make a clear definition of security and to prove that the protocol fulfills those definition under some plausible assumptions.

Assume the cash you withdraw had your name on it. What would that mean? In most cases it wouldn’t mean anything. No-one would be interested in knowing that it was you who bought that pack of chewing gum. You might feel a little bit uncomfortable if you knew that a curious trainee working in the pharmacy can keep track of what medicine you use. If the government can figure out your political viewpoint by monitoring what newspapers you purchase and what events you buy tickets to, you have reason to be really worried.

We often take anonymity for granted. If you purchase a newspaper with cash, it is not possible to trace the purchase back to you by looking at the coins you paid with. If you buy a couple of tokens for the metro, it is not possible to see if two trips were paid by tokens purchased at the same time. The simple reason neither coins nor metro tokens are traceable is that they don't have a serial number.

The reason they don't have a serial number is that their low value don't make them an interesting target for counterfeiter – the cost of producing a fake coin or metro token probably exceeds its value. Now you may argue that these transactions are not at all anonymous – if you go and buy the newspaper in person, anyone can see what you bought. However, the important point here is that it requires considerable resources to track a person that way, and it is impossible to do in an automated way on a large scale.

When the physical coins and metro tokens are replaced with electronic counterparts, the scenario is changed. The cost of copying an electronic coin, which is nothing but a sequence of zeros and ones, is next to nothing. Therefore even low-value coins need some kind of serial number to detect duplicates, and that potentially makes them traceable. One of the challenges when designing protocols for transactions that people assume to be anonymous is to make them anonymous also when performed electronically.

Before we can design anonymous protocols, we must decide what we mean by anonymity. One definition of anonymity is that a transaction cannot be connected to the identity of any involved party. This definition, however, is weaker than the anonymity of real-world transactions, because it does not say anything about connecting transactions. Assume, for example, that you use your electronic coins first to buy a train ticket that is mailed to your home and then to buy a political newsletter.

If the coins are anonymous only in the above sense, the identity of the buyer of the newsletter may still be revealed if the two purchases can be connected. Clearly the latter kind of anonymity is preferable to the former.

If a protocol involves several parties, in the case of electronic coins a customer, a merchant and the bank, we may settle for anonymity only towards the merchant to make the protocol more efficient. In other words, the merchant cannot link two purchases, but once the coin reaches the bank, the bank can see who withdrew the coin. Another concept is **revocable anonymity**. Here some trusted third party (who could, for example, be a judge) can extract the identity from a coin, but otherwise the coin is anonymous towards both the bank and the vendor.

Although anonymity is desirable from the user's point of view, protocols that ensure anonymity tend to be less efficient than non-anonymous protocols. Also from a legal point of view anonymity might be problematic. If electronic coins are achieved through black-mailing or other illegal activities, anonymity works in favor of the criminal.

In an anonymous scheme for electronic coins the bank cannot monitor the flow of coins. It will detect irregularities only after a long period of time (if ever). This may be one reason why the schemes for electronic cash that are in use are non-anonymous.

1.7 Payment Systems

When making purchases, the most common ways to pay for the goods is either by using cash or by using a payment card or check. Cash has the property that it is anonymous and that it is possible to verify that it is valid by just looking at it and without calling the bank. This offline property of cash is important, and very desirable. It reduces communication costs, it makes the scheme more robust since it doesn't require the bank to be available, and it is fast. The merchant can deposit the cash with his bank, use it as change, buy goods, pay salary etc. Unfortunately cash also has the not so nice property that it can be stolen.

A payment card or check, on the other hand, is not itself a proof that the customer has the money to pay. The issuer must be contacted to verify that the customer has the necessary funds, but once the transaction is completed, it cannot be stolen like cash. Since the merchant's name is part of the payment, no-one else can get credited for the transaction. Digital payment systems try to mimic these properties. Systems for digital cash try to keep the anonymity of the customer, possibly with a trusted party that can revoke the anonymity. However, since a digital coin is just a bit-string, it can be copied and spent twice. The most common way to deal with this is to design the system so that the identity of the owner is revealed if the same coin is spent twice. Another solution is to make the system online, but then part of the motivation to use coins is lost.

Systems for digital cash often require that the merchant deposits the cash with the bank after the transaction rather than reuse it.

However, digital cash may also have the useful property that in cannot be stolen while at the merchant, since the merchant's name is part of the transaction. If digital cash does not completely correspond to cash in the real world, payment card transactions are easier to make purely electronic. In many cases this simply means that the physical signature on the receipt is replaced by a digital signature by the cardholder. Here, however, we can ask for more and make payment card transactions anonymous towards the merchant.

The goal then is to design a system such that two transactions cannot be linked by the merchants. The system will still be non-anonymous towards the issuer, since it must be able to charge the correct account. A trivial way to achieve anonymity towards the merchant is to give each cardholder not just one card number, but several one-time numbers. The bank keeps a list of which number belongs to which cardholder, and the cardholder makes sure each number is only used once. Provided that the card numbers are generated randomly, such a system would be anonymous towards the merchants.

1.8 Group Signatures

In this section we discuss a more general approach to the problem of creating anonymous credit cards. We use the concept of **group signatures**. In a group signature scheme, there are **group members** and a **group manager**. Group members can sign documents on behalf of the group, but the only information that someone other than the group manager gets is that **someone** in the group signed the document. The group manager, however, is able to determine the identity of a signer. As the alert reader already has seen, this is exactly what we need to make payment cards anonymous. The group members are the cardholders, and the issuer is the group manager.

When making a payment, the cardholder produces a group signature on the transaction. The merchant verifies that the signature is produced by someone in the group of cardholders, but does not get any additional information. When the transaction is passed on to the card issuer, the issuer, who acts as group manager, extracts the identity of the cardholder to debit the correct account.

The scheme described above with group signatures works for payment cards when there is just one issuer, and every merchant sends all transactions directly to that issuer. In reality this is not the case. There is not just one but several issuers cooperating within a network. Rather than sending the transaction directly to the issuer, the merchant sends it to the network, which routes it to the issuer. The obvious way to solve the problem is to set up a group signature scheme for each issuer. With this solution we lose some anonymity, since the merchant learns the name of the issuer, and in some cases this can give quite a lot of information.

Therefore we would like a variant of group signatures where there are group managers that

only get partial information about the identity of the signer.

More specifically, in the case of payment cards, we need a scheme such that the signature is anonymous to the merchant, the network can see which issuer the card belongs to, and the issuer sees the identity of the cardholder. Naturally this can be generalized so that there are several intermediate group managers that get more and more detailed information about the identity. In this thesis we describe such an extension of group signatures. Because of the hierarchical way information about the identity is revealed, we call the scheme hierarchical group signatures.

1.9 EMV Payment Cards

Still the majority of payment cards are equipped with a magnetic stripe where the cardholder data is encoded. Although a convenient and cheap solution, it has its security problems. The magnetic stripe can be copied and modified, making it a good target for counterfeit and fraud. The transactions made with a magnetic stripe are not digitally signed, making it possible to modify the transaction data after the transaction took place.

One alternative to the magnetic stripe is **smart-cards**. A smart-card is a tiny computer placed on a plastic card. As with any computer, it can store and process data. It can also have some parts of its memory protected from direct access. This is a very useful property to prevent copying and modification of cards.

Since the amount of money lost on fraud by the payment networks is growing, there is an on-going program to switch to smart-cards. The switch is currently in progress, with some issuers already issuing smart-cards, and some still using the magnetic stripe.

With smart-cards, the security is increased considerably. A smart-card cannot be copied or modified the same way a magnetic stripe can. It can hold secret data used only internally by the card. Smart-cards can sign transactions, thus ensuring they are sent to the payment network unmodified. Some smart-cards also contain a private key for authentication purposes. Since the private key is accessible only to the internal smart-card software, such a card cannot be duplicated.

Cardholder data on a smart-card may be digitally signed by the issuer, preventing it from being modified as data on the magnetic stripe.

With the magnetic stripe a cardholder can pay wherever his brand of card is accepted. He doesn't have to worry about who manufactured the terminal or which bank will process the payment, since all magnetic stripe cards and all terminals work according to the same standards. For the switch to smart-cards to be successful, the same interoperability is necessary also for smart-cards. Therefore an international, publicly available standard called EMV has been developed.

1.10. CRYPTOGRAPHY AND LATTICES

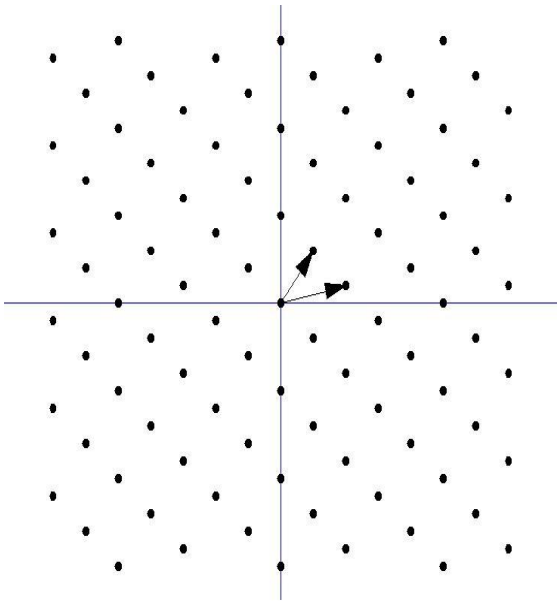

Figure 1.1: A two-dimensional lattice

In this thesis we point out a vulnerability in some EMV cards. Although the EMV standard builds on primitives in which no vulnerabilities are known, we show that certain EMV card configurations are insecure. The vulnerability would allow an attacker to use an EMV card to perform an unlimited number of offline

transactions. EMV does allow for offline transactions, but there is a limit on the maximum number of consecutive offline transactions stored on the card. In Chapter 4 we show how to perform the attack, and also, where it is possible, how to configure a card to protect against the vulnerability.

As we have seen, we need an underlying hard problem to design cryptosystems. One family of such problems are **lattice** problems. A lattice is defined as the set $\{\lambda_1 b_1 + \lambda_2 b_2 + \cdots + \lambda_n b_n\}$ where λ_i are integers and $b_i \in R^n$. Put differently, a lattice is defined by n basis vectors in R^n. The lattice consists of points in R^n (sometimes called **lattice vectors**) generated by adding combinations of the basis vectors with integral coefficients. In Figure 1.1 a basis for a two-dimensional lattice is shown together with the lattice points generated by the lattice.

In SVP, the task is to compute the shortest non-zero vector in a lattice given a basis for the lattice. (The zero vector obviously is the shortest vector for any lattice.) In Figure 1.2 the shortest vector in the two-dimensional lattice is marked, and here we see that in general the shortest vector is not one of the basis vectors, and that the shortest vector is never unique, since if v is a lattice vector, then so is −v.

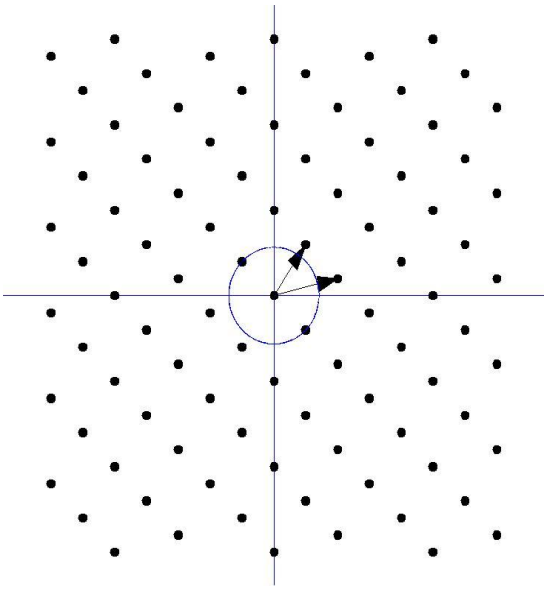

Figure 1.2: The shortest vector in the lattice

Computing a shortest vector in a two-dimensional lattice is not difficult, but in lattices of higher dimension the general consensus is that no algorithm which efficiently solves SVP exists. Now, if we can't expect to find the shortest vector in reasonable time, it is natural to ask if we can find a vector which may not be the shortest, but which isn't too much longer than the shortest.

It turns out that the answer to this question depends on what one means by "not too much longer". It is known that finding a lattice vector that is up to a factor k longer than the shortest is essentially as hard as finding the shortest vector for any constant k. On the other hand there is an efficient algorithm that is known to always give a vector that is at most $2^{n/2}$ times as long as the shortest vector, and that in practice often produces even better results. It is still unknown precisely where the border lies between what can be computed efficiently and what cannot.

Lattice problems have cryptographic applications. It is known that the crypto-system NTRU would be insecure if short vectors could be found in a certain type of lattices. Since the NTRU lattices are of very high dimension, it is believed to be infeasible to find such short vectors. However, the NTRU lattices have a certain structure that could potentially make them weaker. In this thesis we study this structure and show that SVP isn't easier in this type of lattices. Our approach is to show that given an arbitrary lattice Λ_1, it is possible to compute a lattice Λ_2 which has the special structure and lies very close to Λ_1. This is shown in Figure 1.3. Now we can conclude that if SVP were easy in Λ_2, then it would be easy in Λ_1 as well, since a solution to SVP in Λ_2 can be translated into a solution in Λ_1 as well. Therefore the special structure of Λ_2 does not help when solving SVP.

1.11. THESIS OVERVIEW

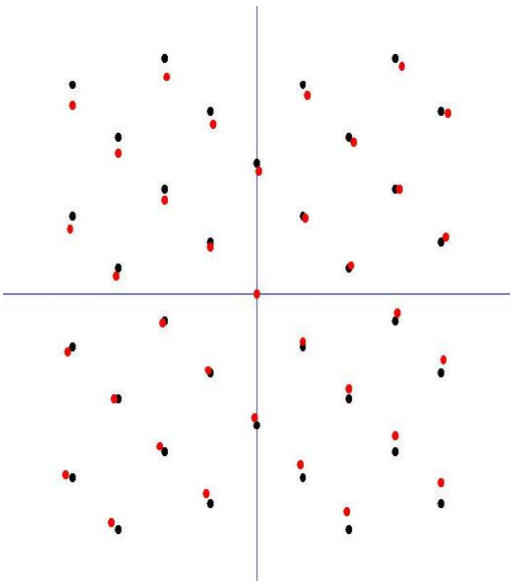

Figure 1.3: Approximating a lattice with another lattice

The thesis is organized as follows. In Chapter 2. We describe a protocol for electronic cash that is designed specifically to be as efficient as possible. In Chapter 3 the protocol for hierchical signatures with proofs of security can be found. Chapter 3 is joint work with Douglas Wikström. In Chapter 4 the weakness of certain EMV payment cards is analyzed. In Chapter 5 we give the full details of the lattice result.

Chapter 2

An Efficient Protocol for Electronic Cash

2.1 Introduction

Today, a large and growing part of payments are made by electronic means, but there is still much room for improvement. Credit cards may be suitable for large amounts, but for small amounts the cost of using a credit card is too high. Also, a credit card is closely tied to the person's identity, and we may want a system where the merchant learns less or even nothing about the identity of the customer. These are the issues to be addressed by electronic cash. The purpose of electronic cash is to give an alternate option for payment which provides some anonymity to the customer, and possibly avoids the need for contacting the bank for every transaction. We present a system for electronic cash that is based on symmetric primitives. The advantage of this is that we get a system where the coins are small and where the cryptographic functions performed by the customer requires little processing power.

Previous Work

A system for electronic cash is usually designed for a situation where the coin is withdrawn from the bank by the customer, transferred from the customer to the merchant (as means of payment) and later deposited by the merchant at the bank. Sometimes it is desirable to have a system where the coins can be transferred between customers in several steps before they are deposited at the bank.

The different security issues that need to be addressed include forgeability (cre-ating a coin that the merchant accepts without performing the withdrawal protocol with the bank first), double spending (making a copy of the coin and spending it twice), and revealing of identity (ability for the bank and the merchant to see who withdrew a coin used in a purchase).

Previously published systems for electronic cash include the system presented by Chaum, Fiat and Naor, which addresses the issues of anonymity and detection of double spenders. Later systems have made improvements. We shown how to make the communication between the merchant and the customer more efficient. In a proposal for how to make the coins divisible is introduced. In the possibility of later revoking the anonymity of the coins is added, which may be desirable for legal reasons. Sander and Ta-Shma present a system where the bank does not have a secret key. Our system is based on the ideas of that system. The similarities and differences between our system and the system introduced by Sander and TaShma is discussed in more detail in section 2.2.

All these systems use asymmetric encryption or non-interactive zero-knowledge proofs to achieve security. The use of asymmetric techniques such as RSA appears to imply that a coin must include numbers of size at least 768 bits, and probably at least 1024 bits. Since a coin often consists of several such numbers, storing the coins on a smart card where the storage is limited is problematic. With non-interactive zero-knowledge proofs, especially when based on general methods, the coins get even larger.

We could of course use a handheld computer to store the coins. This would however make the system less convenient to use and thus less likely to be accepted. The cost of such devices would reduce the likely hood that the system is widely accepted. Therefore we want the emphasize the small coin size of the presented system.

Privacy, Coin Sizes and Efficiency

As mentioned before, in the systems presented so far, anonymity is a major concern. They ensure that neither the merchant nor the bank can identify the owner of a coin. This is achieved either by the use of blind signatures or non-interactive zero-knowledge proofs of knowledge. Both methods generate coins that are "large" (meaning having a size such that a number n of the same size is hard to factor, or that it is hard to find the discrete logarithm modulo n).

We propose a system that is significantly more efficient than the previously published systems, and still provides full anonymity towards the merchant. The system only uses symmetric encryption and computation of hash functions, thus eliminating the need for costly operations like exponentiation. The emphasis is on efficiency – the same functionality can easily be accomplished using public key cryptography, but this would yield much larger coins.

To get some perspective we can compare the size of the coins in our system with previously proposed systems. The system proposed by Ferguson needs five RSA-sized number per coin, giving each coin a size of $5 \cdot 1024 = 5120$ bits, or 640 bytes. The system has reduced this to three numbers of 1024 bits, giving each coin a size of 384 bytes.

2.1. INTRODUCTION

Our Solution

In this chapter we describe a simple and efficient system for electronic cash with provable security properties. The system relies on symmetric encryption technologies rather than asymmetric. This enhances performance, and the system still gives a fair level of anonymity. Another advantage is that the bank does not have a master secret that can be stolen and used for counterfeiting money. We present two variants of the system, one that is completely offline and one that is online. In the online variant, central databases are used to store information that otherwise would be stored on the customer's smart card.

Previously published systems focus on anonymity, both from the merchant and from the bank. They make it impossible for both the merchant and the bank to trace a payment. This un-traceability may be desirable in certain cases, and it is certainly in the customer's interest. It is however far from certain that a bank would want (or accept) that kind of anonymity. There are also law enforcement aspects – if money is used in blackmailing, we would like to be able to trace the money.

The system presented here is semi-anonymous. The merchant cannot trace a payment. In fact, it cannot see whether or not two payments have been made by the same customer. The bank, however, can see the identity of the customer when the merchant deposits his money, just as it would with, for example, a credit card.

By sacrificing anonymity against the bank, we win a lot in coin size. The technique used to avoid the need for a signature on every coin is the use of hash trees as proposed by Merkle [49] and used by Sander and Ta-Shma [63]. The idea is that the bank keeps the coins it has issued in hash trees, where each father is the hash value of the concatenated values its sons. Creating hash trees is a one-way process – given the leaves it is easy to compute the root value, but given only the root value it is infeasible to construct a matching tree (except for the trivial tree consisting of only the root). The roots of the trees are made public, and any coin which has a path leading to a published root is regarded as valid. Since the paths are not secret, it is possible to publish these paths, removing the need for the paths to be stored on the smart card. Also, the correctness of these paths is defined by the fact that they lead to a certified root. This means that the databases containing the paths do not need to be in any way secure or authenticated.

The merchant can himself verify the outcome by comparing the root with the certified roots he has received from the bank. How does this system differ from an ordinary credit card system? When paying with a credit card, the customer have to reveal his identity to the merchant, whereas in the proposed system the customer remains anonymous to the merchant. In the online version the purchase must be verified against a database, but unlike a credit card system, this database does not need to be authenticated. Also, the communication does not have to be secure. A large merchant may even keep a copy of the

database locally, to speed up the processing. The offline version has the obvious advantage that there is no need for an online connection to an external database. An implementation of the scheme is underway.

2.2 Overview of the System

The system uses hash trees to keep track of which coins should be considered valid. In the online variant, the hash trees are distributed to local databases via not necessarily secure lines. In the offline variant, only the bank keeps a copy of the hash trees and the customers keep track of the path of every coin. Only the roots need to be transmitted on an authenticated line.

All participants have agreed on a parameter k, which needs to be even. H is a hash function. $R_a(x)$ is a pseudorandom function with the key a. In practice we can think of H as SHA-1. As pseudorandom function we can use a symmetric cryptosystem (like AES) with a as the key.

The Participants

There are three participants – the customer, the merchant and the bank. The cus-tomer is assumed to have a smart card or similar device with some, but limited, storage and computational capabilities. The customer's identity is id. The cus-tomer's smart card is assumed to have a secret, which we call a. The card also contains a secret key used to identify the customer with some signature scheme. The bank has the corresponding public key. The bank does not have a master secret key. The bank only needs to be able to, in an authenticated way, publish roots of the trees of hash values.

The bank also has a symmetric cryptosystem, whose encryption we call E, and whose decryption is called D. The merchant has no secret. The merchant has to get the root of the hash tree the bank has published in a secure way, and it has to have access to a database which contains the hash tree, although this access does not have to be authenticated.

The Protocol

The protocol consists of three steps – withdrawal, payment and deposition. In the withdrawal phase, the customer receives coins from the bank and the bank charges the customer's bank account. In the payment phase, the customer transfers coins to the merchant. In the deposition phase, the merchant deposits the money with the bank, and the bank credits the amount to the merchant's account.

Withdrawal

When withdrawing money from his account, a secure channel is set up between the customer and the bank. The customer first identifies himself to the bank in some way (possibly using his private key). A withdrawal of a coin then proceeds as follows

1. The bank generates a serial number, s, and sends c = E(s, id) to the customer.
2. The customer commutes $z_i = H(R_a(c, i))$ for i = 1, . . . , k, and sends these signed to the bank.
3. The banks allocates a position in the next hash tree, and sends the path to this position (as a **{**0, 1**}**-string) to the customer. Later, when the bank actu-ally builds the tree, the customer's coin is inserted as a leaf in the allocated position.

This is described in Figure 2.1. We call the pair (c, a) a coin.

After the protocol has finished, the customer has the coin represented by c, and knows k values that hash to values in the hash tree. Also, this information is not known to anyone but the customer.

Note that the use of a pseudo-random function is only to save space. The same security would be achieved if the customer in step 2 generated k values, say $b_1, b_2, \ldots, b_k$ and sent $H(b_i)$ to the bank. In such a setup, the customer would need to store the values k $b_1, b_2, \ldots, b_k$, whereas he in the current setup does not need to store any extra information apart from c.

An alternative to having c as an encryption of (s, id) would be to have c as a unique random number, and to have the bank store the pair (c, id) in a private database. The current setup avoids the need for an extra database with sensitive information. Such a setup would, on the other hand, remove the need for a secret key for the bank.

Payment

In the payment phase, the customer sends the public part of the coin c together with the challenges $\mathbf{h}z_i$ **i** and the path through the hash tree to the merchant. In the online variant, the merchant turns to the database to get the necessary path. In the offline variant, the user has the path and sends it to the merchant. All the merchant needs to check is that one end of the path contains the hash value of c and the challenges and that the other end contains the root it has received on a secure channel. Together with the path, the merchant receives the z_i's.

The procedure is as follows. From the start the customer has a coin (c, a).

1. The customer sends c, $\mathbf{h}z_i\mathbf{i}$ and the path to the merchant.
2. The merchant verifies c and $\mathbf{h}z_i\mathbf{i}$ with the database. The merchant picks numbers
3. $b_1, \ldots, b_k$ at random from **{**0, 1**}** under the constraint that $^{\mathbf{P}}\, b_i = k/2$.
3. The customer computes $y_i = R_a(c, i)$ for each i such that $b_i = 1$ and sends the result to the merchant.
4. The merchant accepts if $H(y_i) = z_i$ for every y_i. If this is the case, the merchant stores these values.
5.

Deposition

The merchant deposits the coin at the bank by sending the y_i's together with c. The bank decrypts c and marks the coin as used in its database. Should the coin already be marked as used, it checks which y_i's were used in the previous transaction. If the same y_i's were used in that transaction, it is assumed that the merchant is trying to deposit the same coin twice, and the merchant's account is only credited once. If the y_i's are different, the customer has tried to double spend his coin. It is then easy for the bank to retrieve the identity of the double spender, since this information is stored in the coin.

Maintaining the Hash Tree

The idea of the hash tree is the idea presented after a certain period of time t, say one minute, the bank creates a hash tree where each leaf consists of a coin c issued during that period and the corresponding challenges z_i. The bank forms the tree and publishes the root. It then gives the path to the customers that have withdrawn coins during that last period.

2.3 Analysis of the System :

For the analysis we want to prove two things. First we need to prove that for honest participants, the system gives a fair result. Then we need to prove that the system is secure. The first part follows directly from the construction of the system and that $D(E(s, id)) = (s, id)$ and that H and R_a are deterministic.

Proof of Security

The setting for the system is that the customer trusts the bank for anonymity and for providing fair coins. Both the merchant and the customer trust the roots of the hash trees. The bank does not need to trust anyone. The customer does not need to trust the merchant and the merchant does not need to trust the customer. No one needs to trust the hash trees provided by the third party database providers.

By neglible probability we mean a probability lower than 1/p(n) for any poly-nomial p and large enough security parameter n. The relevant security parameters are the number of challenges, the length of the customer's private key, the length of the bank's private key and the output length of the hash function.

We start by giving definitions of the tools we need. Since these definitions are standard definitions, we only give informal descriptions.

Definition 2.3.1 (Collision resistant hash function).

A keyed hash function $H_a : A \rightarrow \{0, 1\}^n$ with key a is called collision resistant if a polynomial time (in |a|) probabilistic Turing machine has negligible probability of finding x, y ∈ A, x 6= y such that H(x) = H(y).

Definition 2.3.2 (Pseudorandom functions).

A function $R_a : A \rightarrow B$, where a is a parameter, is called pseudorandom if it indistinguishable from a uniformly distributed functions f : A → B to a Turing machine that runs in time polynomial in |a|.

Definition 2.3.3 (Semantic security).

We call a symmetric cryptosystem (E, D) semantically secure if no polynomial time probabilistic Turing machine T can distinguish between $E(m_0)$ and $E(m_1)$ with non-negligible probability, even if T is allowed to pick m_0 and m_1 itself.

Definition 2.3.4 (Detect identity).

We say that two merchants can detect identity if they can play the following game and be successful with a probability non-negligibly higher than 1/2.

2.4. SOME PRACTICAL DETAILS

there exists an a such that $y_i^1 = R_a(c^1, i)$ and $y_i^2 = R_a(c^2, i)$ for every challenge y_i. Otherwise the output bit is 1 with probability q.

We want to use the oracle to violate the pseudo randomness of R_a. Assume we have a family of functions F given as a black box and we want to decide whether it is pseudorandom or random.

We want to create an algorithm A (which uses O as an oracle) that with non-negligible probability outputs 1 if F is pseudorandom and 0 otherwise.

Consider the following four distributions of coins, where a and b are two different keys for the pseudorandom function, and B is a random function.

1. Both coins are withdrawn using R_a.
2. Both coins are withdrawn using B.
3. The first coin is withdrawn using R_a and the second coin using B.
4. The first coin is withdrawn using R_a and the second coin using R_b.

Let p_i be the probability the oracle O outputs 1 on in data from distribution i. From the assumption that O can detect identity it follows that $p_1 - p_4$ is non-negligible. This implies that one of $p_1 - p_2$, $p_2 - p_3$ or $p_3 - p_4$ is non-negligible. We show how to implement A in each of these cases

- If p_1 **6**= p_2 we create both coins using F and use this as input to O.
- If p_2 **6**= p_3 we create one coin using F and the other using a random function and execute O with this as input.
- If p_3 **6**= p_4 we create one coin using R_a and the other using F and use this as input to O.

Since for at least one of these inequalities the difference is non-negligible, our al-gorithm is able to distinguish the family of pseudorandom functions from random functions, which was assumed to be infeasible. This is a contradiction which proves the theorem.

Choosing the Challenges and Stealing of Coins

As we have seen, the way the challenges are chosen is very important for the de-tection of double spending. The simplest would be to choose the challenges at random. The probability of the challenges being identical is then very low. One could, however, imagine legal problems if the bank tries to prove in court that two merchants have collaborated to perform fraud against the bank by both de-positing the same coin. The merchants can of course argue that they happened to accidently pick the same random challenge (maybe due to bad (pseudo-) random number generators).

With this setup, a merchant A could steal coins from merchant B. If A steals the hard-disk containing the only copy of B's coins, the theft would never be discovered by the bank. Several practical precautions are of course possible, but we will see that with a slight modification of the protocol, stealing coins is impossible.

One way of choosing challenges would be to have the challenge consist of a hash value of, say, the merchant's identity, the time, the amount and possibly other parameters.

The problem with this setup is that since k cannot be very large, it would be possible for the customer to find a collision. He can then choose to spend his coin where he has found the collision, and hence get exactly the same challenge. The bank would not be able to decide whether it is the merchants or the customer that is guilty of the fraud.

Since the only property of randomness we used is that we have low probability of the same challenge, we could instead allocate a certain interval of challenges to each merchant (or rather to each terminal accepting payments). We would then have a counter in every terminal to ensure that no set of challenges is used twice. If a terminal runs out of queries, it would notify the bank which would assign a new interval. If we assume that we want each interval to consist of 10000 queries, and we want 10^9 such intervals, k = 50 would be enough.

A feature of this setup is that a merchant A cannot steal a coin from another merchant B, since this stolen coin has a challenge that does not correspond to a valid challenge of A, and the bank detects that the coin has been stolen. The same is true if a merchant eavesdrops a purchase or a deposition of a coin.

The Coin Databases

Another important part of the payment system is the coin databases. Since every coin issued is to be stored in these databases, they need quite large storage capabilities. We can note that the information in these databases is in no way sensitive. It does not have to be authenticated or secure, and with the cost of storage medium being low (and only getting lower), this is not such big a problem as it may seem. On the other hand, smart card memory is expensive, so it is a good thing to save smart card memory in favor of hard disk space.

For each coin only the hash value needs to be stored in the database. To further reduce storage need we can design the system so that the coins are issued in a distributed manner. For every entity with a certain number of customers (e.g., every city), the coins would be combined into a hash tree. These local hash trees would be joined into national hash trees, which in turn would be joined on an international level.

With this setup most databases can be designed to hold only information on coins issued in the same city or country, and during the last period of time, say a month. There would probably be a few complete databases that would be contacted in case the coin could not be found in the local database, and these would need to contain every coin that has been issued.

2.5 CONCLUSIONS

If we assume that a person spends 100 coins per day, one million customers would spend approximately 3×10^9 coins per month. Since every coin needs 160 bit in the database (assuming we use SHA-1 as hash function) the total storage need would be 60 GB, which is far from infeasible, especially as the security requirements of the database are low. The central databases would need to hold more information.

Values of Coins

The system as described here assumes all coins have the same value. It is easy to adopt the system to handle coins of different values. When the bank issues the coin, it can simply add the value of the coin to the leaf in the database. The value does

not have to be added explicitly – it suffices to have the hash value stored in the database include the value of the coin.

The Size of a Coin

We now calculate the size of a coin (c, a). The secret a is common to all coins, so the size of it does not have to be counted. The variable c consists of the encryption of a serial number s and the identity id. If we want the system to scale for world-wide use, we need to reserve, say, 64 bits for the identity. We can then use 64 bits for the serial number, and still be able to fit the coin in 128 bits. We only need the serial number to be unique per customer. Using a symmetrical cryptosystem with a key-length of 128 bits, we get an output of 128 bits.

In the online variant, we need to store the path in the hash tree. If we assume that the system globally has 10^{10} users and every user uses 100 coins per day for 100 years, no path needs more than 60 bits. This gives a total coin size of less than 200 bits, or 25 bytes, which is a major improvement compared to the asymmetric cash systems described in the introduction.

In the offline variant, the path needs to be stored by the customer. This is more than can be stored on a smart card, which means we need some means of secondary storage.

Assuming a tranfer rate of 9600 bps between the card and the terminal, the most costly phase of the payment phase, transfer of the responses to the challenges, requires less than half a second, assuming that 25 challenges have to be answered. We have presented a system for electronic cash that is both practical and provably secure. The privacy properties are such that banks are likely to accept the system, and the system still protects the customer's identity against the merchants. The coins are small enough to fit on smart cards.

Chapter 3

Hierarchical Group Signatures

3.1 Introduction

Consider the notion of group signatures introduced by Chaum and van Heyst [21]. A group member can compute a signature that to an outsider reveals nothing about the signer's identity except that he is a member of the group. On the other hand the group manager can always reveal the identity of the signer.

An application for group signatures is anonymous credit cards. The cardholder wishes to preserve his privacy when he pays a merchant for goods, i.e., he is in-terested in unlinkability of payments. The bank must obviously be able to extract the identity of a cardholder from a payment or at least an identifier for an account, to be able to debit the account. To avoid fraud, the bank, the merchant, and the cardholder all require that a cardholder cannot pay for goods without holding a valid card. To solve the problem using group signatures we let the bank be the group manager and the cardholders be signers. A cardholder signs a transaction and hands it to the merchant. The merchant then hands the signed transaction to the bank, which debits the cardholder and credits the merchant. Since signatures are unlikeable, the merchant learns nothing about the cardholder's identity. The bank on the other hand can always extract the cardholder's identity from a valid signature and debit the correct account.

The above scenario is somewhat simplified since normally there are many banks that issue cards of the same brand and which are processed through the same payment network. The payment network normally works as an administrator and routes transactions to several independent banks. Thus, the merchant hands a payment to the payment network which hands the payment to the issuing bank. We could apply group signatures here as well by making the payment network act as the group manager. The network would then send the extracted identity to the issuing bank. Another option is to set up several independent group signatures schemes, one for each issuer. In the first approach, the payment network learns the identity of the customer, and in the second approach the merchant learns which bank issued the customer's card. A better solution would reveal nothing except what is absolutely necessary to each party. The merchant needs to be convinced that the credit card is valid, the payment network must be able to route the payment to the correct card issuer and the issuer must be able to determine the identity of the cardholder.

A solution that comes to mind is to use ordinary group signatures with the modification that the customer encrypts his identity with his bank's public key. Then we have the problem of showing to the merchant that this encryption contains valid information. However, the customer cannot reveal the public key of the bank to the merchant, making such a proof far from trivial.

In this chapter we introduce and investigate the notion of **hierarchical group signatures**. These can be employed to solve the above problem. When using a hierarchical group signature scheme there is not one single group manager.

Instead there are several group managers organized in a tree, i.e., each group manager either manages a group of signers or a group of group managers. In the original notion the group manager can always identify the signer of a message, but nobody else can distinguish between signatures by different signers. The corresponding property for hierarchical group signatures is more complicated. If a manager directly manages a group of signers, it can identify all the signers that it manages, but the signatures of all other signers are indistinguishable to it. This corresponds directly to the original notion. If a manager manages a group of managers, it cannot identify the signer, but it can identify the manager directly below it which (perhaps indirectly) manages the signer. Thus, a manager that does not manage signers directly get only partial information on the identity of the signer.

When we use hierarchical group signatures to construct anonymous credit cards for the more realistic setting we let the payment network be the root manager that manages a set of group managers, i.e., the issuing banks, and we let the cardholders be signers. The credit card application also demonstrates what kind of responsibility model is likely to be used with a hierarchical group signature scheme. With a valid signature on a transaction, the merchant has a valid demand on the payment network. If the payment network has a signature that can be shown to belong to a certain bank, the network has a valid demand on that bank. Thus, it is in the network's interest to open the signatures it receives from merchants, and it is in the issuing banks' interest to open the signatures they receive from the network.

3.2 HIERARCHICAL GROUP SIGNATURES :

In Section 3.5 we consider the issue of existence of such primes. We use QR_N to denote the subgroup of squares in Z^*_N , i.e., the quadratic residues. We write $\emptyset$ to denote both the empty set and the empty string.

We say that a distribution ensemble $\mathbf{D} = \{D_\kappa\}$ is efficiently sampleable if there exists a polynomial time Turing machine T_D that on input 1^κ outputs a random sample distributed according to D_κ. All adversaries in this chapter are modeled as polynomial time Turing machines with **non-uniform** auxiliary advice string. We denote the set of such adversaries by PPT^*.

A public-key cryptosystem is said to be **CCA2-secure** if it is infeasible for an attacker to determine which one of two messages of his choice that a given cryptotext is the encryption of, even if the attacker has access to a decryption oracle both before the choice is made and after the cryptotext is received [60]

In Section 3.2 we formalize the notion of hierarchical group signatures and give definitions of security. We also briefly discuss why it is not trivial to transform a non-hierarchical group signature scheme into a hierarchical scheme. In Section 3.3 we introduce the concept of cross-indistinguishability, which we use in both the general construction and the explicit construction. Our construction under general assumptions is presented in Section 3.4 and in Section 3.5 we give the explicit construction. The zero-knowledge proofs used in Section 3.5 can be found in Section 3.6. Finally in Sections 3.7 and 3.8 we discuss possible modifications and extensions of the current scheme.

Contributions

We introduce and formalize the notion of **hierarchical group signatures**. We give a construction that is provably secure under the existence of a trapdoor permutation family. As part of our investigations we introduce a new property of cryptosystems, which we call cross-indistinguishability. This property may be of independent in-terest.

Then we consider how a practical hierarchical group signature scheme can be constructed under specific complexity assumptions. We show that by a careful selection of primitives one can construct a relatively practical hierarchical group signature scheme that is provably secure under the DDH assumption and the strong RSA assumption in the random oracle model. For reasonable security parameters a few hundred exponentiations are required to produce a signature.

3.3.CROSS-INDISTINGUISHABILITY

of such a scheme are complex and involves many subtle issues, e.g. should all group managers (indirect and direct) of a signer get information on its identity, or should the signer decide on a path from a root and only reveal information to group managers along this path? Although we believe that the techniques we use for our construction would be useful also for this type of scheme we do not investigate such schemes further.

On Constructing Hierarchical Group Signatures

All known group signatures are based on the idea that the signer encrypts a secret of some sort using the group manager's public key, and then proves that the resulting cryptotext is on this special form. The security of the cryptosystem used implies anonymity, since no adversary can distinguish crypto texts of two distinct messages if they are encrypted using the **same** public key.

Suppose we wish to generalize this approach to construct a hierarchical group signature scheme. In the hierarchical setting protecting the identity of the signer implies protecting the identity of the group managers along the path of to the signer. On the other hand these group managers (and nobody else) must be able to extract partial knowledge on the identity on the identity of the signer. Thus, it seems that hierarchical group signatures must somehow contain embedded crypto texts. To ensure anonymity, signatures with embedded crypto texts corresponding to distinct public keys must be indistinguishable, since otherwise the crypto texts embedded in a signature would reveal information on the identity of the signer.

This type of in distinguishability does not follow from the in distinguishability of a cryptosystem. We say that a cryptosystem that has this property is cross-indistinguishable. This property is investigated in detail in Section 3.3 below.

On the other hand, to ensure traceability, the signer must prove that a signature contains the identity of the signer encrypted with public keys corresponding to the path to the signer. In principle this is not a problem, since there is a non-interactive zero-knowledge proof system for any language in **NP**, but the details must be resolved. It is far from obvious how to construct a practical proof system.

It turns out that the cryptosystem we use must not only be indistinguishable (semantically secure), but it must also have an incomparable security property which we call **cross-in distinguishability**.

3.4 A CONSTRUCTION UNDER GENERAL ASSUMPTIONS:

In this section we show how hierarchical group signatures can be constructed under general assumptions. Our focus is on feasibility and conceptual simplicity. We prove the following theorem.

Theorem 3.4.1. If there exists a family of trapdoor permutations, then there exists a secure hierarchical group signature scheme.

To prove the theorem we construct a hierarchical group signature scheme by augmenting the group signature scheme of [7] with additional crypto texts and a non-interactive zero-knowledge proof.

Assumptions and Primitives Used

Before we give our construction we review some constructions and results on which our construction is based.

Group Signature Scheme

The first building block we need is a group signature scheme secure under the assumption that trapdoor permutations exists. As shown by Bellare et al. such a scheme exists.

Theorem 3.4.2 (cf. [7]). If there exists a family of trapdoor permutations, then there exists a secure group signature scheme GS = (GKg, GSig, GVf , Open).

Public Key Cryptosystem

The probabilistic cryptosystem of Goldwasser and Micali [34] is indistinguishable, but we are not aware of any proof of cross-indistinguishability. We prove that their construction is also cross-indistinguishable, but first we recall their construction.

Their construction is based on the existence of non-approximable trapdoor predicates. This concept can be captured in modern terminology as

follows. A family of trapdoor permutations is a triple of polynomial time algorithms **F** = (Gen, Eval, Invert). The instance generator Gen(1^{κ}) outputs a description f of a permutation of **{**0, 1**}**$^{\kappa}$ and a trapdoor f^{-1}. The evaluation algorithm Eval(1^{κ}, f, x) evaluates the permutation on input x **∈ {**0, 1**}**$^{\kappa}$. The the corresponding inversion algorithm Invert(1^{κ}, f^{-1}, y) evaluates the inverse permutation on input y **∈ {**0, 1**}**$^{\kappa}$. We abuse notation and write f (x) and f^{-1}(y) for the evaluation of the permutation and inverse permutation as described above.

The last requirement on the family of trapdoor permutations is that it must be infeasible for any A **∈** PPT* given f and y = f (x), where x **∈ {**0, 1**}**$^{\kappa}$, to compute x = f^{-1}(y). A hard-core bit for **F** is a family of functions **B** = **{**B_{κ} : **{**0, 1**}** $^{\kappa}$ → {0, 1**}}** such that it is infeasible to compute B_k(x), given only f and f (x) for a random x **∈ {**0, 1**}**$^{\kappa}$. Goldreich and Levin [33] show how to construct a family of trapdoor permutations **F** with a hard-core bit **B** from any family of trapdoor permutations.

The cryptosystem **GM** = (GMKg, E, D) of Goldwasser and Micali [34] using **F** and **B** can be defined as follows (using modern terminology). The key generator GMKg(1^{κ}) simply outputs (pk , sk) = (f, f^{-1}) = Gen(1^{κ}).

3.5 CONCLUSION

We have introduced and formalized the notion of hierarchical group signatures and given two constructions. The first construction is provably secure under general assumptions, whereas the second is provably secure under the DDH assumption, the strong RSA assumption and the 4-Cunningham chain assumption in the random oracle model.

Although the latter construction is practical, i.e., it can be implemented and run on modern workstations, it is still relatively slow. Thus, an interesting open problem is to find more efficient constructions of hierarchical group signatures.

cards based on M/Chip cannot be configured in the proposed way, and are therefore always susceptible to the attack. Here the only solution is to move to (more expensive) DDA cards.

One possibility to solve the problem is the change the EMV specification so that a terminal always goes online when a non-DDA EMV card is used. Although the consequence is that issuers using low-cost card cannot benefit from the advantages of offline transaction, from a security perspective this approach would be the most efficient.

CHAPTER - 4

On the Security of Non-RSA EMV

Payment Cards :

4.1 Introduction

A large part of today's electronic purchases are made with different kinds of pay-ment cards. The majority of the cards used today have a magnetic stripe where the card data is stored. Over the last years, card skimming, where the content of the magnetic stripe is copied, has become a major problem. The countermeasure is to move from the magnetic stripe to smart-cards where the data is stored on a chip instead. To make sure also smart-card based payment cards will have the same global acceptance as the magnetic stripe, Europay, MasterCard and Visa have together developed the EMV specification.

The preparations for moving from payment cards based on magnetic stripe to smart-card based cards have been going on for more than ten years. Some card issuers have already converted their card base to EMV smart-cards, and more are about to make the switch.

The base for EMV smart-cards is the EMV specifications [29], which define the protocol between the card and the terminal. Payment organizations, in particular Visa and MasterCard, have developed their own extensions to the EMV specifica-tions [48, 47, 73].

In this chapter we will examine a potential problem in the configuration of an EMV card. In particular we will show how to avoid this problem with a card based on Visa's VSDC specification, and that it cannot be avoided when using MasterCard's M/Chip specification. EMV specifies two possible security levels for cards, Static Data Authentication (SDA) and Dynamic Data Authentication (DDA). The difference lies in that DDA cards must support RSA, whereas an SDA card does not. The issue we discuss in this chapter relates only to SDA cards.

4.2 SMART-CARDS

A smart-card is a tiny computer with its own CPU and storage. The data stored on the card cannot be read or written directly but only through certain functions. This means that a smart-card may have keys that can be used for encryption but cannot be read in clear. Another possilibity is to have a PIN that must be entered before certain functions can be used. More information about smart-cards can be found in [38].

The fact that the data on the card can only be accessed through predefined functions means that we can define data to be public when it can be accessed in clear and private when it is used only for internal processing by the card. When analyzing protocols involving smart-cards, a reasonable security model is to assume the data can only be accessed and modified using the predefined functions. The weakness analyzed in this chapter follows that security model.

4.3 THE EMV SPECIFICATION

The EMV specification describes in detail the data flow between the card and the terminal during a transaction. The outcome of an execution the protocol is one of the following

1. Transaction is approved offline.
2. Transaction is denied offline.
3. ransaction is sent to the issuer for online authorization Since most transaction are either approved offline or sent online, we will consider only these two cases here. The principle is that a transaction can be approved offline only if both the card and the terminal agrees on it, but is sent online if at least one of them requests it.

Both Visa and MasterCard have written their own extensions to EMV. Here they define which of the public EMV parameters that can be used, and also what the internal behavior of the card should be. Visa calls their application VSDC [73]. MasterCard has published two separate documents, one giving the external interface in the form of minimum requirements [47] and one defining the internal behavior by describing the

application M/Chip [48]. However, also MasterCard is moving towards a unified document giving both internal and external details.

Card-Issuer Authentication

During a transaction, the card may generate one or two MACs. These MACs are generated with a symmetric key known only to the issuer and the card. Therefore the MAC can only be verified by the issuer and not by the terminal or the acquirer.

Card Configuration

SDA and DDA.

EMV gives the option of using low-cost cards without RSA capabilities as well as more expensive RSA enabled cards. Cards without RSA capabilities support only Static Data Authentication (SDA) whereas cards with RSA support can handle also

Dynamic Data Authentication (DDA).

For both SDA and DDA, the issuer receives a certificate from the payment organization. The issuer certificate and the issuer public key, IPK, are stored publicly on the card. For SDA, the issuer signs a set of card parameters of his choice with his private key and places the signature on the card. The signature is called Signed Static Application Data, SSAD. In the case of DDA the card is given its own RSA key pair. The card private key is stored internally on the card, but the card public key is signed by the issuer. (Even if the card supports DDA, an SDA signature is usually still put on the card.)

Card parameters

Apart from the keys and certificates mentioned above, several parameters describing under which circumstances to allow offline transactions are stored on the card. In this thesis we are only interested in one parameter, namely Lower Consecutive Offline Limit (LCOL). The LCOL gives the number of transactions that can be performed offline, i.e., without contacting the issuer.

Also the parameters Application Transaction Counter (ATC) and Last Online Application Transaction Counter (LATC) are stored on the card. The ATC con-tains the number of transaction the card has performed and the LATC holds the index of latest transaction that executed online. They are both initialized to zero.

Symmetric keys

When a card is issued, the issuer generates a symmetric key that is stored on the card and used to generate MACs transaction messages.[2] The key is also stored by issuer, but not disclosed to the merchants or the acquirers.

4.4 THE PROBLEM

In this section we will describe the potential problem, and also how to avoid it wh ere possible.

Making a Pure Online Card

In many cases, it is desirable to have a card that can only function online. There are two ways to achieve this:

- Set the LCOL to zero. This way the terminal will always make a transaction go online.
- Make the internal risk analysis of the card such that it always makes the decision to go online, regardless of what the terminal's decision is.

The most obvious reason to make a card online-only is to make sure the card-holder does not spend money he does not have. However, for an SDA card there is also another reason. Since the MAC cannot be verified offline, someone might copy the card, keeping the original SDA signature, but replace the symmetric key.

Then a terminal would accept the card (since the SDA signature is valid), and when (and if) the issuer detects that the MAC is invalid, it is already too late. The essence of the attack described here

is to copy the card and modify the copy in such a way that it will allow offline transactions.

Copying an SDA Card

If we assume that the hardware is secure, the adversary can copy all the public data on a card, but not the internal data. Also, when the card is copied, he can modify data that has not been signed by the issuer, but if he changes the data included in the SDA signature the card will not be accepted. When a card is copied the adversary can change the internal behavior of the card by replacing the original program code by his own.

Making an Online Card Work Offline

As mentioned above, there are two ways to make a card online-only. Here we will discuss different attack scenarios depending on what method is used.

If the LCOL parameter is set to zero, then it may either be signed or not be signed with the SDA signature. If it is not signed, the adversary can simply copy the card and modify the LCOL to contain a non-zero value. The card will be accepted offline, since the SDA signature is still valid. He will not be able to copy the symmetric key, so the copy cannot be used online, but as long as the card only is used offline, it will work. In case the LCOL is signed, it cannot be changed and the attack does not work.

If the LCOL is not present on the card, but the internal risk analysis of the card is used to make all transaction go online, then the attack is a little bit different. When the adversary copies the card, he replaces the card application with an application that always accepts to make the transaction offline. Also here he cannot copy the symmtric key, but he will be able to use the card offline.

Note that for any of these attacks, the adversary only needs access to the card for a few seconds so that he can read all the public data. Since the commands for doing this are standardized, any card-reader could be used for this.

Copying an Offline-enabled Card

If the card has LCOL non-zero, but not in the SDA signature, the adversary can of course use a similar method to get an arbitrary number of offline transactions (with invalid MACs, making it impossible to tie the transaction to the card). However, even if LCOL is signed, the adversary can issue an attack similar to those described above. He can copy all the parameters on the card, but modify the card application to that it always responds that no offline transactions have been performed prior to the current. That way the terminal will always accept to make the transaction offline (since the number of offline transactions is lower than the LCOL) and the issuer will not be able to detect that the MAC is invalid.

Protecting Against the Attack

As we can see, the only way to make the card secure against the proposed attack is to set the LCOL to zero and include it in the SDA signature. In other words, there is no way of making a secure offline SDA card.[3]

However, the specifications for M/Chip [48] (used for MasterCard) don't allow the use of LCOL, leaving only card-based risk analysis for making a card online-only. As we have seen, such an approach is always susceptible to the attack by modifying the application. (The M/Chip specifications do define the LCOL, but only as private parameter used internally by the card.)

[3]This attack does not work for DDA cards.

It can be noted that inclusion of LCOL in the data signed with SDA is not in the published recommendations. One step for reducing the potential threat is to update the recommendations to include the LCOL and also note that it should be set to zero.

Conclusions and Recommendations

We have demonstrated how EMV cards with a certain configuration can be at-tacked, and we

have also pointed out how to configure a card correctly to avoid this attack. We have seen that cards based on M/Chip cannot be configured in the proposed way, and are therefore always susceptible to the attack. Here the only solution is to move to (more expensive) DDA cards.

One possibility to solve the problem is the change the EMV specification so that a terminal always goes online when a non-DDA EMV card is used. Although the consequence is that issuers using low-cost card cannot benefit from the advantages of offline transaction, from a security perspective this approach would be the most efficient.

Chapter 5

Lattices With Many Cycles Are Dense

5.1 Introduction

The interest in the computational complexity of lattice problems started in the beginning of the 1980s, when van Emde Boas published the first **NP**-completeness result for lattice problems [71]. Several hardness results for different variants of this problems and for different subsets of lattices have followed. One such way of classifying lattices is according to the cycle structure of Abelian group Z^n/Λ, which is the main focus of this chapter. Previous results on the complexity of lattice problems that either explicitly or implicitly consider lattices with a certain cycle structure include [1, 13, 56, 67].

The group Z^n/Λ is finite if $\Lambda \subseteq Z^n$ and full-dimensional. One way to visu-alize this group is to divide Z^n into the parallelepipeds spanned by a basis and consider two points equivalent if they lie in the same position in their respective parellelepipeds. In Figure 5.1 one such equivalence class of points is shown. Note how this can be considered a generalization of reduction modulo an integer over Z. It is easy to see that Z^n/Λ is a group under addition, and since addition is commutative, the group is abelian.

As with any abelian group, it is isomorphic to the cartesian product of cyclic groups. By writing the cycle lengths in increasing order so that the length of cycle i divides the length of cycle i + 1, we get a unique representation. For example, instead of writing $Z_3 \times Z_5$ we write Z_{15}, and instead of $Z_2 \times Z_3 \times Z_3$ we write $Z_3 \times Z_6$.

There are two reasons to study the hardness of certain lattice problems in dif-ferent subclasses of lattices rather than for general lattices. The first reason is purely theoretical – it gives us a better understanding of how the computational complexity of lattice problems behaves if we restrict ourselves to certain lattice classes. The second reason is more practical – most hardness results are worst-case results for general lattices.

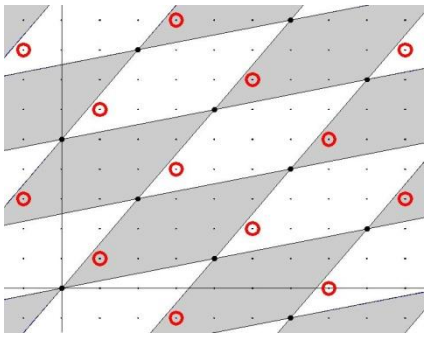

Figure 5.1: Points that are equivalent modulo a lattice certain structural properties. It would be desired to have results that show that these properties cannot be used to solve lattice problem more efficiently.

The first result on the cycle structure was published by Paz and Schnorr [56]. In their paper it is shown that any lattice can be approximated arbitrarily well by a lattice with one cycle. In other words, the lattices with one cycle form a hard core. On the other hand, the lattices Cai and Nerurkar [13] prove to be hard in the improved version of Ajtai [1] have up to n/c cycles. Although the results are diff erent in nature (the latter is not an **NP**-hardness result), it is interesting to note that they give hardness results for lattices with diff erent cycle structure. This gives rise to the question of the role of the cycle structure in the complexity of lattice problems.

The influence of the cycle structure on the hardness of lattice problems has practical implications.

For some crypto systems (e.g., NTRU [37]) there are attacks based on finding short vectors in certain lattices. The lattices used in some of these attacks have a cycle structure that diff ers from the cycle structure of the lattices that previously have been shown to be **NP**-hard.

Since a lattice with n cycles always can be transformed into a lattice with fewer cycles by a simple rescaling, the maximum number of cycles that is meaningful to analyze is $n - 1$. Trolin showed that the exact version SVP under the max-

norm is **NP**-complete for n-dimensional lattices with n − 1 cycles of equal length [67].

In this chapter we investigate the importance of the cycle structure further. Our main result is a polynomial-time transformation that with arbitrary precision approximates any n-dimensional lattice with a lattice that has n − 1 cycles of equal length, showing that these lattices form a hard core. A consequence of this is that short vectors and close vectors cannot be computed more efficiently in this class of lattices than in general lattices, except possibly for a polynomial factor. As our transformation only changes the size of the coordinates of the basis vectors and not the dimension of the lattice, the transformation is rather tight.

5.2. BACKGROUND

We also give a theorem showing a connection between the subdeterminants of a lattice and its Smith Normal Form. An i-minor of B is an i × i matrix formed by taking i rows and i columns of B.

Theorem 5.2.4. Let B be an integral square matrix. Then the diagonal elements of the Smith Normal Form, $s_1, s_2, \ldots, s_n$ can be computed as where d_i is gcd of the determinants of all i-minors of B, and $d_0 = 1$.

Although this method of computing the Smith Normal Form and hence the cycle structure is quite inefficient (we need consider all the i-minors, not only the principal), it turns out to be useful in certain proofs in this chapter. There are other, more efficient methods to compute the Smith Normal Form [39].

Another way to describe the number of cycles of a lattice is to use a different representation of the lattice, namely as a set of modular equations. Every lattice can be described in this way.

Theorem 5.2.5. Let $\Lambda \subseteq Z^n$ be a lattice. Then there exist n-dimensional vectors $a_1, a_2, \ldots, a_m$ and integers $b_1, b_2, \ldots, b_m$, $b_i > 1$, such that

$\Lambda = \{x : ha_1, xi \equiv 0 \bmod b_1 \wedge ha_2, xi \equiv 0 \bmod b_2 \wedge \ldots \wedge ha_m, xi \equiv 0 \bmod b_m\}$.

The essence of this theorem is that any lattice can be expressed as a system of modular linear equations whose solutions form the lattice.

The connection to the cycle structure is that the number of nontrivial cycles is m, and the length of cycle i is b_i, provided that the system of equations has been reduced to minimize the number of equations and that the gcd of the coefficients and the modulus is 1 in each equation.

In the transformations we approximate lattices in Z^n with lattices in Q^n. The standard definition of cycle structure cannot be applied to general lattices in Q^n. Since multiplication by a constant does not affect lattice problems such as SVP and CVP, we will define the cycle structure of a lattice $\Lambda \subset Q^n$ as the cycle structure of $k\Lambda$, where k is the smallest integer such that $k\Lambda \subseteq Z^n$.

5.3 The Approximation

Let $\Lambda \subseteq Z^n$ be an arbitrary lattice. To adapt this into a lattice with n − 1 cycles that is arbitrarily close to the original lattice we go through the following five steps:

1. Inflate the lattice by a factor k and perturb to achieve a lattice with Hermite Normal Form of a certain form.
2. Reduce the sublattice spanned by the first n−1 vectors of the Hermite Normal Form using the LLL algorithm.
3. Factor the partly reduced basis matrix into two matrices, where the second has its determinant equal to one.
4. Multiply the two matrices to get a basis for an (n − 1)-cyclic lattice that is close to the original lattice.
5. Perform modifications to the first matrix to give it n−1 cycles of equal length.

5.5. CONCLUSIONS

Lemma 5.4.3. Let ($\Lambda \subseteq Z^n$, $y \in Z^n$) be an instance of CVP such that $0 \leq y_i < \det(\Lambda)$. Then $x \in \Lambda$ is a

solution if and only if $_k^1\sigma_{\Lambda,\varepsilon}(x)$ is a solution of the instance

$_k^1\sigma_\varepsilon(\Lambda)$, $_k^1\sigma_{\Lambda,\varepsilon}(y)$– for k and ε^{-1} polynomial in det(Λ) and n.

Proof. The lemma follows directly from Theorem 5.4.1. Using the two lemmas, we can construct the reduction by first reducing the target vector modulo det(Λ) and then apply the transformation with the appropriate value of ε.

Obviously the same technique can be used to achieve a similar result for SVP. The following lemma follows directly from the above lemmas.

Lemma 5.4.4. Let $\Lambda \subseteq Z^n$ be an instance of SVP. Then $x \in \Lambda$ is a solution if and only if $_k^1\sigma_{\Lambda,\varepsilon}(x)$ is a solution of the instance $_k^1\sigma_\varepsilon(\Lambda)$ for k and ε^{-1} polynomial in det(Λ) and n.

From this we can conclude that the inapproximability results for SVP and CVP from [41] and [27] hold also for lattices with n − 1 cycles.

Theorem 5.4.5. SVP in ℓ_p-norm is NP-hard to approximate within any constant factor for n-dimensional lattices with n − 1 non-trivial cycles of equal length.

Theorem 5.4.6. There exist constants c_p such that CVP is NP-hard to approximate within $n^{\log\log n}$ in ℓ_p-norm for n-dimensional lattices with n − 1 non-trivial cycles of equal length.

We have constructed a transformation that given an n-dimensional lattice of any cycle structure produces a lattice with n − 1 cycles that is arbitrarily close to the original lattice. This closes the question of whether SVP and CVP can be easier to solve in lattices with many cycles. Using the presented result, such a solution would give a solution for the general case that is at most a polynomial factor slower in running time. Also the known inapproximability results for SVP and CVP extend to lattices with n − 1 cycles.

By previous results, we know that any lattice can be approximated arbitrarily well by a cyclic lattice, and hence that SVP and CVP cannot be easier to solve in cyclic lattices than in general lattices, except possibly for a polynomial factor. We now have the two extremes, for one cycle and for n − 1 cycles.

From the results by Ajtai and the improvement by others we have a hardness result also for lattices with n/c cycles. Together with our result this gives evidence for the general hypothesis that the cycle structure have little importance in deciding the hardness of SVP and CVP in a certain lattice.

Although it does seem likely that also lattices with m non-trivial cycles form a hard core for $2 \le m \le n - 2$, we don't have a proof for this. The current proof does not easily extend to these cycle structures. Since our method relies on inflating the lattice by a factor d^t to get a lattice with determinant d^{nt+1} and then making changes to achieve m cycles, the length of each cycle is $d^{(nt+1)/m}$. Naturally t must be chosen so that (nt + 1)/m is an integer. In our case, we achieve this by setting t = n − 2 and m = n − 1. Since the value of t would depend on m and for certain relations between m and n no such t exists at all, our method cannot directly be generalized to create any cycle structure where the non-trivial cycles have equal length.

Even if a transformation into m cycles of equal length for $1 \le m \le n - 1$ were found it would still be an open question whether other cycle structures, where the cycles have diff erent lengths, remain easy. Still the current result seems to be a strong indication that the cycle structure does not play an important role for the computational complexity of lattice problems.

the lattice by a factor d^t to get a lattice with determinant d^{nt+1} and then making changes to achieve m cycles, the length of each cycle is $d^{(nt+1)/m}$. Naturally t must be chosen so that (nt + 1)/m is an integer. In our case, we achieve this by setting t = n − 2 and m = n − 1. Since the value of t would depend on m and for certain relations

between m and n no such t exists at all, our method cannot directly be generalized to create any cycle structure where the non-trivial cycles have equal length.

Even if a transformation into m cycles of equal length for $1 \leq m \leq n - 1$ were found it would still be an open question whether other cycle structures, where the cycles have diff erent lengths, remain easy. Still the current result seems to be a strong indication that the cycle structure does not play an important role for the computational complexity of lattice problems.

Bibliography

1. M. Ajtai. Generating hard instances of lattice problems. In 28th ACM Sym-posium on the Theory of Computing (STOC), pages 99–108. ACM Press, 1996.
2. M. Ajtai. The shortest vector problem in ℓ_2 is NP-hard for randomized reduc-tions. In 30th ACM Symposium on the Theory of Computing (STOC), pages 10–19. ACM Press, 1998.
3. G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In Advances in Cryptology – CRYPTO 2000, volume 1880 of Lecture Notes in Computer Science, pages 255–270. Springer Verlag, 2000.
4. G. Ateniese and G. Tsudik. Some open issues and directions in group signa-tures. In Financial Cryptography '99, volume 1648 of Lecture Notes in Com-puter Science, pages 196–211. Springer Verlag, 1999.
5. L. Babai. Trading group theory for randomness. In 17th ACM Symposium on the Theory of Computing (STOC), pages 421–429. ACM Press, 1985.
6. Bellare and O. Goldreich. On defining proofs of knowledge. In Advances in Cryptology – CRYPTO'92, volume 740 of Lecture Notes in Computer Science, pages 390–420. Springer Verlag, 1992.
7. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on gen-eral assumptions. In Advances in Cryptology – EUROCRYPT 2003, volume 2656 of Lecture Notes in Computer Science, pages 614–629. Springer Verlag, 2003.
8. M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In 20th ACM Symposium on the Theory of Computing (STOC), pages 103–118. ACM Press, 1988.
9. F. Boudot. Efficient proofs that a committed number lies in an interval. In Advances in Cryptology – EUROCRYPT 2000, volume 1807 of Lecture Notes in Computer Science, pages 431–444. Springer Verlag, 2000.
10. F. Boudot and J. Traoré. Efficient publicly veriable secret sharing schemes with fast or delayed recovery. In 2nd International Conference on Information and Communication Security (ICICS), volume 1726 of Lecture Notes in Computer Science, pages 87–102. Springer Verlag, 1999.
11. S. Brands. Untraceable off -line cash in wallets with observers. In Advances in Cryptology – CRYPTO'93, volume 773 of Lecture Notes in Computer Science, pages 302–318. Springer Verlag, 1994.
12. E. Brickell, P. Gemmell, and D. Kravitz. Tracing extensions to anonymous cash and the making of anonymous change. In 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 457–466. ACM Press, 1995.
13. J-Y. Cai and A. Nerurkar. An improved worst-case to average-case connection for lattice problems. In 38th IEEE Symposium on ACM Symposium on the Theory of Computing (STOC), pages 468–477. IEEE Computer Society Press, 1997.
14. J. Camenisch. Efficient and generalized group signature. In Advances in Cryptology – EUROCRYPT'97, volume 1233 of Lecture Notes in Computer Science, pages 465–479. Springer Verlag, 1997.

15. J. Camenisch and M. Michels. A group signature scheme with improved effi-ency. In Advances in Cryptology – ASIACRYPT'98, volume 1514 of Lecture Notes in Computer Science, pages 160–174. Springer Verlag, 1999.
16. J. Camenisch and M. Michels. Separability and efficiency for generic group signature schemes. In Advances in Cryptology – CRYPTO'99, volume 1666 of Lecture Notes in Computer Science, pages 413–430. Springer Verlag, 1999.
17. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In Advances in Cryptology – CRYPTO'97, volume 1294 of Lecture Notes in Computer Science, pages 410–424. Springer Verlag, 1997.
18. R. Canetti, O. Goldreich, and S. Halevi. The random oracle model revisited. In 30th ACM Symposium on the Theory of Computing (STOC), pages 209–218. ACM Press, 1998.
19. D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In Advances in Cryptology – CRYPTO'88, volume 403 of Lecture Notes in Computer Science, pages 319–327. Springer Verlag, 1990.
20. D. Chaum, E. van Heijst, and B. Pfitzmann. Cryptographically strong undeni-able signatures, unconditionally secure for the signer. In Advances in Crypto-logy – CRYPTO'91, volume 576 of Lecture Notes in Computer Science, pages 470–484. Springer Verlag, 1991.
21. D. Chaum and E. van Heyst. Group signatures. In Advances in Cryptology – EUROCRYPT'91, volume 547 of Lecture Notes in Computer Science, pages 257–265. Springer Verlag, 1991.
22. L. Chen and T.P. Pedersen. New group signature schemes. In Advances in Cryptology – EUROCRYPT'94, volume 950 of Lecture Notes in Computer Sci-ence, pages 171–181. Springer Verlag, 1994.
23. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Advances in Cryptology – CRYPTO'94, volume 839 of Lecture Notes in Computer Science, pages 174– 187. Springer Verlag, 1994.
24. R. Cramer and V. Shoup. A practical public key cryptosystem provably se-cure against adaptive chosen ciphertext attack. In Advances in Cryptology – CRYPTO'98, volume 1462 of Lecture Notes in Computer Science, pages 13–25. Springer Verlag, 1998.
25. R. Cramer and V. Shoup. Signature schemes based on the strong RSA as-sumption. In 6th ACM Conference on Computer and Communications Security (CCS), pages 46–51. ACM Press, 1999.
26. D. Chaum and E. van Heyst. Group signatures. In Advances in Cryptology – EUROCRYPT'91, volume 547 of Lecture Notes in Computer Science, pages 257–265. Springer Verlag, 1991.
27. L. Chen and T.P. Pedersen. New group signature schemes. In Advances in Cryptology – EUROCRYPT'94, volume 950 of Lecture Notes in Computer Sci-ence, pages 171–181. Springer Verlag, 1994.
28. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Advances in Cryptology – CRYPTO'94, volume 839 of Lecture Notes in Computer Science, pages 174– 187. Springer Verlag, 1994.
29. R. Cramer and V. Shoup. A practical public key cryptosystem provably se-cure against adaptive chosen ciphertext attack. In Advances in Cryptology – CRYPTO'98, volume 1462 of Lecture Notes in Computer Science, pages 13–25. Springer Verlag, 1998.
30. R. Cramer and V. Shoup. Signature schemes based on the strong RSA as-sumption. In 6th ACM Conference on Computer and Communications Security (CCS), pages 46–51. ACM Press, 1999.

AUTHOR PROFILE

Dr. Daruri Venugopal Received the Bachelors and Masters degrees from Osmania University, in 1995 and 1997, respectively. He done his M.Phil Mathematics from Algappa University in the year 2003. He done his M.Tech Computer Science Engineering from JRN Deemed University. He Done his Doctorate in Computer Science Engineering. He has over 110 Research Paper to his Credit. He is Editorial Board Member and Reviewer for four Reputed International Journals in Mathematics & Computer Science Areas. He is a Advisory Board Member of Reputed Technical Institutions. He is a Life Member of ISTE, He is a Recognized Ph.D Supervisor in the Areas of Mathematics and also in Computer Science and Network Engineering. Presently working as Professor in Siddhartha Institute of Technology and Sciences, Ghatkesar, Hyderabad.