Zero Knowledge Protocols and Small Systems

Hannu A. Aronsson
Department of Computer Science
Helsinki University of Technology
haa@cs.hut.fi

Table of Contents

---

1. Introduction

Zero-knowledge protocols allow identification, key exchange and other basic cryptographic operations to be implemented without leaking any secret information during the conversation and with smaller computational

requirements than using comparable public key protocols. Thus Zero-knowledge protocols seem very attractive especially in smart card and embedded applications.

There is quite a lot written about zero-knowledge protocols in theory, but not so much practical down-to-earth material is available even though zero-knowledge techniques have been used in many applications.

Some of the practical aspects of zero-knowledge protocols and related issues are discussed, in the mind-set of minimalistic practical environments. The hardware technology used in these environments is described, and resulting real-world practical problems are related to zero-knowledge protocols.

A very lightweight zero knowledge protocol is outlined and its possible uses and cryptographic strengths and weaknesses are analyzed.

2. Zero-knowledge Protocol Basics

Zero-knowledge protocols, as their name says, are cryptographic protocols which do not reveal the information or secret itself during the protocol, or to any eavesdropper. They have some very interesting properties, e.g. as the secret itself (e.g. your identity) is not transferred to the verifying party, they cannot try to masquerade as you to any third party.

Although Zero-knowledge protocols look a bit unusual, most usual cryptographic problems can be solved by using them, as well as with public key cryptography. For some applications, like key exchange (for later normal cheap and fast symmetric encryption on the communications link) or proving mutual identities, zero-knowledge protocols can in many occasions be a very good and suitable solution.

2.1 The parties in a Zero-knowledge protocol

The following *people* appear in zero-knowledge protocols:

Peggy the Prover
: Peggy has some information that she wants to prove to Victor, but she doesn't want to tell the secret itself to Victor.

Victor the Verifier
: Victor asks Peggy a series of questions, trying to find out if Peggy really knows the secret or not. Victor does not learn anything of the secret itself, even if he would cheat or not adhere to the protocol.

Eve the Eavesdropper
: Eve is listening to the conversation between Peggy and Victor. A good zero-knowledge protocol also makes sure that any third-party will not learn a thing about the secret, and will not even be able to replay it for anyone else later to convince them.

Maggie the Malice
: Maggie is listening to the protocol traffic and maliciously sending extra messages and modifying or destroying messages. The protocol must be tamper-resistant to this kind of activity.

These names are used widely in this paper and elsewhere in the public key cryptography literature.

2.2 Zero-knowledge terminology

The *secret* means some piece of information, be it a password, the private key of a public key cryptosystem, a solution to some mathematical problem or a set of credentials. With Zero-knowledge protocols, the prover can convince the verifier that she is in possession of the knowledge, the secret, without revealing the secret itself, unlike e.g. normal username-password queries.

Accreditation means the building of confidence in each iteration of the protocol. If in one step of a zero-knowledge protocol, the chance of an impostor being able to provide the answer is 1 in 2, the chances of her passing an entire conversation are 2^-(number of accreditation rounds).

Often the prover will offer a *problem* (i.e. particular numeric values for a generic hard-to-solve mathematical problem, e.g. factoring extremely large numbers, which are products of large primes) to the verifier, which will ask for one of the 2 or more possible solutions. If the prover knows the real solution to the hard mathematical problem, she is able to provide any of the solutions asked for. If she doesn't know the real solution, she can not provide all of the possible solutions, and if the verifier asks for one of the other solutions, she is unable to provide it, and will be found out.

Cut-and-choose protocols work in the way, that one failure means the failure of the whole protocol (i.e. that the prover is not legitimate), but you can keep working on the protocol as long as you want, if the prover is legitimate. After you reach the level of confidence you need without being cut off, the protocol is successful.

2.3 Features of Zero-knowledge Protocols

Zero-knowledge protocols can be described as cryptographic protocols having the following special features:

The verifier can not learn anything from the protocol

> The Verifier does not learn anything from the protocol, that he could not learn all by himself, without the prover. This is the central zero-knowledge concept, i.e. No knowledge (zero amount of knowledge) is transferred.
> Without this feature, the protocol would be called a minimum-disclosure protocol, i.e. zero-knowledge protocols require that absolutely no information can be leaked in any case.

The prover can not cheat the verifier

> If Peggy doesn't know the secret, she can only succeed with a great amount of good luck. After several rounds of the protocol, the odds of an impostor passing as legitimate can be made as low as necessary. The protocols are also cut and choose, i.e. the first time the prover fails, Victor knows Peggy is not legitimate. So, with each round of the protocol, the certainty gets better and better.
> The protocols can be made to work even if the odds of a guess passing are high, you just need more rounds in the protocol.

The verifier can't cheat the prover

> Victor can't get any information out of the protocol, even if he does not follow the protocol. The only thing Victor can do is to convince himself that Peggy knows the secret. The Prover will always reveal only one solution of many to any one problem, never all of them which would allow finding out the secret itself.

The verifier can not pretend to be the prover to any third party

> Because no information can leak from Peggy to Victor, Victor can't try to masquerade as Peggy to any outside third party. With some of these protocols, a man-in-the-middle attack is possible, though,

> meaning that someone can relay the traffic from the true Peggy and try to convince another Victor that he, the perpetrator, is Peggy.
> Also, if the verifier records the conversation between him and the prover, that recording can't be used to convince any third party. It looks the same as a faked conversation (e.g. where the verifier and prover agreed beforehand which requests the verifier will choose).

2.4 Modes of Operation

The zero-knowledge protocols can be used in three main modes.

Interactive, where Peggy and Victor interactively go through the protocol, building up the certainty piece by piece.

Parallel, where Peggy creates a number of problems and Victor asks for a number of solutions at a time. This can be used to bring down the number of interactive messages with a slow-response-time connection.

Off line, where Peggy creates a number of problems, and then uses a cryptographically strong one-way hash function on the data and the set of problems to play the role of Victor, to select a random solution wanted for each problem. She then appends these solutions to the message. This mode can be used for *digital signatures* .

2.5 Computational Requirements

Many sources claim that Zero-Knowledge protocols have lighter computational requirements than e.g. public key protocols. The usual claim is that Zero-Knowledge protocols can achieve the same results than public key protocols with one to two orders of magnitude less (1/10 1/100) computing power.

A typical implementation might require 20 30 modular multiplications (with full-length bit strings) that can be optimized to 10 20 with precalculation. This is much faster than RSA.

The memory requirements seem to be about equal - to have very high security with Zero-knowledge protocols, you will need very long keys and numbers, so in memory terms, the requirements may not be very different.

Iterative Relatively Light Transactions

The main incentive in using Zero-Knowledge Protocols instead of common public key protocols, are the lighter computational requirements. Sometimes you may also need the special properties of Zero-Knowledge protocols, but for most common tasks, you could also use public key protocols with equal success, so the choice boils down to the computational requirements:

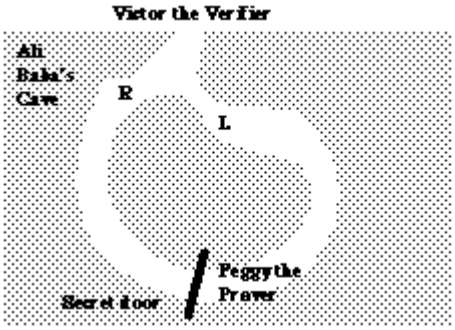
> Zero-Knowledge mechanisms let you split the protocol into an iterative process of lighter transactions, instead of one heavy transaction.

It seems possible to create a protocol with (many) very light iterative rounds, minimizing the computation and memory requirements of the protocol at any one time. More on this as we go along.

Example: Ali Baba's Cave

Consider the example of a circular variety of Ali Baba's cave, with a secret door that can be opened by a password. Peggy knows the password of the door, and wants to convince Victor that she knows it, but doesn't

want Victor to know the password itself.



They work as follows:

- Peggy goes into a random branch of the cave, which Victor doesn't know standing outside the cave
- Victor comes into the cave, and calls out a random branch of the cave (left or right), where Peggy should come out
- If Peggy knows the secret password, she can come out the right way every time, opening and passing the secret door with the password if necessary. If Peggy doesn't know the password, she has a 50% chance of initially going into the wrong branch, and as she is not able to pass the secret door, Victor can call her bluff.

They repeat this as many times as needed to convince Victor. If Victor will be happy with a 1 in 1024 chance of Peggy not knowing the password, they need 10 repetitions (2^10 = 1024).

This example also demonstrates another feature of zero-knowledge protocols: Now Victor is convinced that Peggy knows the secret password, but he cannot convince anyone else himself, as he doesn't know the secret!

Let's say that Victor would videotape the operation. But that recording can't be used to convince anyone else, as it looks just the same as a faked videotape, where a mischievous verifier and the prover agreed in advance which passage the prover should come out each time.

So, Victor can't even convince others, just himself, about Peggy knowing the secret. Absolutely no information flowed to Victor in the protocol.

Example: Rubik's Cube

The more complex protocols can be illustrated by a non-computer example about solving Rubik's cube. Suppose that Peggy knows how to solve the cube and wants to convince Victor about her skill, but doesn't want to show Victor how to actually solve the cube.

Victor shows Peggy a scrambled Rubik's cube and asks for a proof of her skill to solve it. Peggy shows Victor a new, scrambled Rubik's cube. Victor can then choose to ask either

- for Peggy to show how to move from the new scrambled position to the original scrambled position, or
- for Peggy to show how to move from the new scrambled position to the solved position.

If Peggy knows how to go from the original position to the new scrambled position to the solution, she can answer either of these requests. If she does not know how to solve the cube, she could choose to be able to present one of the two choices that victor might ask for, but not both.

Knowing both parts of the solution means that you can solve the cube, i.e. you know the secret. This is why

Peggy must always present a differently scrambled cube in each round, because if Victor gets both answers to a single problem, he gains the knowledge of how to solve the cube in that case.

3. Zero-Knowledge Protocol Theory

This paper, in it's practical approach doesn't cover the theoretical background in a lot of detail. Readers interested in the exact heavy theory behind zero-knowledge issues are directed to the comprehensive references, e.g. Schneier's book Applied Cryptography [3].

The textbook protocols work with relatively theoretical mathematics, and looking from a practical perspective it's not always so clear how they can be applied in real life applications.

Like other cryptographic protocols, zero-knowledge protocols base themselves on the usual crypto math, e.g. modulo calculation, discrete mathematics, extremely large numbers (hundreds or thousands of bits), etc.

3.1 Cryptographic Strength

The cryptographic strength of zero-knowledge protocols is based on a few hard-to-solve problems, e.g:

- The problem of solving discrete logarithms for large numbers (hundreds of bits)
- The problem of knowing if a number is a square mod *n* or not, if you don't know the factors of n
- The problem of factoring large numbers that are products of two or more large (hundreds of bits) primes

The security of zero-knowledge protocols seems to rest on the same foundation as many other, well researched cryptosystems, so the security of the systems can be quite easily evaluated.

Other, but only theoretically interesting, impractical problems are used in the theoretical protocols.

3.2 Contingency plans for protocol failure?

The same problems are used as the security base of public key protocols, so if an easy solution to any of these problems is found, large groups of cryptographic protocols become obsolete overnight (over a microsecond, you'd better think).

Surprisingly, I haven't seen a designed-in feature in any cryptosystem to handle the case of the cryptographic bottom of the system falling out, e.g. a planned course of action in the system for that case.

A lot of your data may need secure retention for many years. Some people say that NSA is 10 years ahead of others in cryptography and cryptanalysis; think about what everybody may be able to do in 10 years?

We don't know if these problems will be broken ever. To play safe with the ever-increasing computation power, you should at least choose keys that are long enough in your time perspective.

In this paper, though, we look at the practical application of quite short keys (minimizing memory requirements) and try to supplement the weak security of that approach with time limits and other practical techniques.

3.3 Which Problems can be Used in Zero-knowledge Protocols?

This point is mainly theoretical, but gives a scope on what can be used as the problem (to be proved in each

round) in these protocols.

If one-way functions exist, any problem in NP has a zero-knowledge proof. The existence of one-way functions is the weakest assumption for strong encryption's existence [2].

The digital signature protocols rely on the quality and randomness of one-way hash functions.

Computational Limitations

Peggy must be polynomially limited in time. If Peggy would be all-powerful, it would not make much sense for her to try and demonstrate her knowledge, as she could calculate it in every case. The very lightweight protocol described in this paper falls victim in this regard - with short keys a relatively-powerful adversary might be able to cheat. Practical measures will be used to minimize this threat.

4. The Zero-Knowledge Protocols Themselves

In this chapter, we look at existing and proposed zero-knowledge protocols and the typical ways these protocols work, and what kind of basic building blocks are used to build these protocols. Many of the basic ideas can be used for several different kinds of protocols.

4.1 Proof of Knowledge

If Peggy has non-negligible chance of making Victor accept, then Peggy can also compute the secret, from which knowledge is being proved.

The Ali Baba's Cave -example is an non-computer example of a zero-knowledge proof of knowledge. One problem that is often used in zero-knowledge proof-of-knowledge protocols is the knowledge of a discrete logarithm of some large number.

4.2 Proof of Identity, Identification

The protocol must ensure, that nobody can masquerade as Peggy or Victor for any other third party. Also, we want to avoid man-in-the-middle attacks, as described in the Chess Grand master Problem:

The Chess Grand master Problem

Let's say that Alice would like to masquerade as a world championship level chess grand master.

She could set-up a two-room game, inviting the top two players to play with her. She would place each one in a separate room, and then play exactly the move the other chess master did to the other and vice versa. Both masters would think they're playing against Alice, even if they're really playing against each other. She would at least appear to be a strong player if not winning. She doesn't even have to know the rules of the game, chess.

This man-in-the-middle attack can be used against some of the Zero-knowledge proof of identity protocols. One counterattack to the man-in-the-middle attack is imposing time limits for the replies, in the hope that there is not enough time for the man in the middle to relay the communications.

The so-called *Mafia fraud* does the same, intercepting electronic payment messages to cheat a person proving his identity when paying at a cheap restaurant into "proving" his identity elsewhere to buy e.g. expensive

jewels for the criminals.

There is also the potential problem of *multiple identities* , if the system does not guarantee that each person has only one identity. Alice could create several identities, and commit crimes using some identity only once, which could never be traced back to her.

Also, it might be possible to rent or *sell identities*, i.e. Alice could sell her private key to someone else, who could then masquerade as her. This works, because what is being actually proved is that you know some piece of information (a bunch of bits), not really who you are. This is also a more generic problem.

4.3 Feige-Fiat-Shamir Proof of Identity

This is the best-known zero-knowledge *proof of identity protocol* . The history of this protocol has a fun sidetrack, when NSA (in USA) tried to stop the authors (non-americans who did the work outside USA) from publishing their work.

A simplified version of this protocol works as follows (we present the full protocol here to give a taste of what most of these protocols look like. The rest will be handled in a more overviewing fashion):

- Precalculation: An arbitrator generates a random modulus *n* (512-1024 bits) which is the product of two large primes.The arbitrator generates a public and private key pair for Peggy, by choosing a number, *v*, which is a quadratic residue mod n (i.e. *x^2 = v mod n* has a solution, and *v^-1 mod n* exists). This number, *v* , is the public key. The private key is then the smallest *s* for which *s = sqrt(1/v) mod n*.

- The identification protocol proceeds then as follows: Peggy picks a random number *r* where *r<n*. She then computes *x = r^2 mod n* and sends it to Victor.
- Victor sends Peggy a random bit, *b*.

- If the bit is 0, Peggy sends Victor *r*. If the bit is 1, she sends *y = r * s mod n*.

- If the bit was 0, Victor verifies that *x = r^2 mod n*, proving that Peggy knows *sqrt(x)*. If the bit was 1, he verifies that *x = y^2 * v mod n*, proving that Peggy knows *sqrt(x/v)*.

If an impersonator tries to pass for Peggy, she can either pick *r* such that she can reply if Victor sends a 0 or 1 bit, but she cannot prepare for both cases, so she will be found out with a 50% chance each round.

Victor can't try to masquerade as Peggy to another verifier, as the bit Victor randomly sent to Peggy earlier has only a 50% chance of being the same as the second verifier will ask for.

In this protocol, Peggy should never reuse an *r*. If she did, Victor could send her the other random bit, and collect a set of both responses. Then, if he had enough of these, he could try to impersonate Peggy to an outsider.

This protocol can be implemented in a parallel fashion, making the public and private keys be a set of quadratic residues mod n, etc. Then you can do as many rounds in parallel as you have keys in the set, speeding up the protocol (but with larger memory requirements) and needing fewer messages.

4.4 Guillou-Quisquater Proof of Identity

This protocol is more suited to smart card applications, as it tries to keep the size of each accreditation (i.e., each round) to a minimum. But it does require about 3 times the computational power of the Fiat-Shamir

protocol.

This protocol requires that the prover have the following: A bit string of credentials *J* (card ID, validity, bank account number, ...) used as the public key, and application-specific pieces of public information, an exponent *v* and a modulus *n* , which is the product of two secret primes. The private key *B* is calculated so that *JB^v = 1 (mod n)*. The protocol goes as follows:

- Peggy sends Victor her credentials *J*. She has to prove that the *J* are her credentials: She picks a random number *r* , so that *1 < r < n 1* . She sends *T = r^v (mod n)* to Victor.
- Victor picks a random number *d*, so that *0 < d < v 1*, which he sends to Peggy.
- Peggy computes *D = rB^d (mod n)* and sends it to Victor.
- Victor computes *T'=D^v * J^d (mod n)*. If *T = T' (mod n)* , then the authentication succeeds.

4.5 Schnorr's Mutual Authentication

Schnorr's authentication and identification scheme is similar to the abovementioned protocols. It can also be used for digital signatures, when replacing Victor with a one-way hash function.

4.6 Key Exchange

A protocol can be devised where RSA public key -based identification is combined with zero-knowledge properties and key exchange (for a session key). Victor can encrypt a random number with Peggy's public key, and if Peggy can decrypt it with her private key, she is identified. A one-way hash function is used to hide the returned decryption so that Victor can't use Peggy to decrypt any message he wants. Victor can only check that the output of the hash function matches what Peggy sent him. A session key can be placed in the random number in some bits that are never sent in clear over the channel [2].

4.7 Digital Signatures

Most of the zero-knowledge protocols can be turned into a digital signature protocol, if Victor is replaced by a cryptographically secure one-way hash function.

Peggy can create a number of problems, use the one-way hash function as a virtual Victor (which will randomly request one or the other solution to each problem, if the hash function is cryptographically good) and provide those answers.

As input to the one-way hash function, the message and the problems presented to Victor are used. This way, neither the message or the problems can be changed without making the signature void. The output of a good crypto-grade one-way hash function is completely random and unpredictable, so Peggy can't try to tweak the inputs to the hash until she gets suitable values permitting forgery.

The receivers can calculate the hash function themselves, check that the correct solutions were provided and that the solutions were valid. If so, then the signature can be considered valid.

Fiat-Shamir Digital Signature Protocol

Victor takes a vacation, and is replaced by a one-way hash function. Nobody notices, and then Peggy finds out that she is able to make digital signatures. I.e., the basic idea of turning a zero-knowledge interactive

protocol into a off-line signature protocol is used [3], p.294.

Guillou-Quisquater Digital Signature Protocol

This scheme uses the Guillou-Quisquater identification protocol, but Peggy substitutes Hash(message, T) for Victor's bit choices [3], p.298.

With the Guillou-Quisquater signature scheme you can do multiple signatures easier than having everyone sign the document separately: The basic idea is that both signers compute their own *J* and *B* values, which are used together to build the signature.

Schnorr Digital Signature Protocol

As usual, the signature is formed by using a good cryptographically secure one-way hash function as Victor's random selection bits [3], p.303.

4.8 Theoretical Protocols

Hamiltonian cycles of large graphs

The Hamiltonian cycle for a graph is a path through the graph that passes every node exactly once (i.e. a solution to the traveling salesman problem). For an extremely large graph, this is very hard (hard enough) to calculate.

Using this to build a zero-knowledge protocol, the prover's secret is the Hamiltonian cycle of a graph.

The prover gives the verifier a permuted version of the original graph, and the verifier can ask for either

- prove that the graph is a permutation of the original graph, or
- show the Hamiltonian path for the permuted graph.

As usual, one of these can be calculated easily from the original data, but to know both, to be able to respond to both possible requests, requires knowledge of the secret, i.e. the Hamiltonian path of the graph. Of course Peggy must use a different permuted graph in each round, as she should never give both solutions to the same problem to Victor.

This protocol is theoretical because of the requirement for the graph to be extremely large, and the large memory and message size requirements it has.

Isomorphic Graphs

A protocol can also be built on top of the problem that for large graphs, it is very expensive to calculate a graph that is isomorphic to two other known graphs [3], p. 86.

4.9 Esoteric Protocols

Proving ability to break RSA

Peggy knows someone's RSA private key, and wants to prove this knowledge to Victor, but does not want to reveal the key, nor decrypt any message with the key. This can be solved by agreeing on two suitable random

numbers (e.g. with a coin-flipping protocol) and encrypting a random message using the secret key and one of there random numbers, from which Victor can check that the private key used was valid, but doesn't gain any knowledge of they key, nor use Peggy to decrypt any message [3], p. 403.

5. Zero-knowledge Protocols in Practice

In this part, we consider the practical aspects of zero-knowledge protocols. The presented textbook protocols are still relatively heavy to calculate.

Applying these protocols to the real world can be challenging, especially given the constraints of embedded computing. The promise of lighter calculations than public key protocols makes studying this very interesting and potentially very useful.

Let's look at these topics in more detail.

5.1 Real Computational and Memory Requirements

The textbook protocols still require calculations on long keys (in the order of hundreds or over a thousand bits) in their textbook usage.

So the computational requirements are still pretty heavy for small-scale applications. The theoretical protocols are clearly too heavy for many minimalistic embedded solutions. Anyway, the promise of a lighter-weight protocol for many common needs, is worth looking into.

If the application permits using conventional symmetric algorithms (e.g. DES), those are still greatly lighter to calculate. There are other security problems in using symmetrical cryptography in e.g. smart cards, which are discussed in more detail below.

If you can only replace one big and very expensive but definitive public key transaction with a series of big and expensive rounds of zero-knowledge transactions, it might not be worth the trouble.

Here is a summary of the requirements of different cryptographic protocol families and their calculation and memory requirements:

Protocol Family	Message Size	Protocol Iterations	Amount of Calculation	Memory Requirements
Zero-knowledge	large	many	large	large
Public-key	large	one	very large	large
Symmetric	small	one	small	small

There is no clear choice for all applications. Especially in small environments, the available computing power and memory is often a limiting factor in the selection of cryptographic techniques.

5.2 Useful Applications

Smart-card applications are often mentioned as good places to use zero-knowledge protocols, because they are lighter to compute than the usual public key protocols.

You would think that it would be good if your electronic cash card, your medical information card, intelligent key and lock systems, etc had real security to protect the information on them, the access to them, and your

identity.

Many embedded and most smart card systems could use some degree of real security.

Examples of Bad Smart Card Solutions

There are many examples of overly simplistic, and thus easily breakable smart card systems. Let's look at some examples where real security might have made a difference.

Because of the lack of computing power and good algorithms, most satellite reception decryption cards only use very simple cryptographic techniques, such as feedback shift registers etc. Most of these systems have been broken and pirate cards (and pirated pirate cards) are quite widely distributed.

Some digital cash experiments rely on shared secrets or keys stored on the cards (e.g. a symmetric algorithm such as DES is used), which requires that the manufacture of the cards is tightly controlled and the system is still vulnerable to a serious reverse engineering attack.

Embedded and Smart Card Environments

The smart card and embedded environments are very *small* in memory and computation resources. In this regard they are radically different from desktop computing, where computing power and memory are abundant. A typical smart card or embedded system might only have several tens or hundreds bytes of work RAM memory and several kilobytes of program memory. Computationally, cheapest embedded systems max out at around 10 to 15 MIPS (of instruction working on 8-bit data).

5.3 Breaking Smart Cards, A Reality Check

The technology to read protected memory or reverse-engineer smart card CPU's and memories is surprisingly good and it is often quite easy to do.

Even some of the so-called security CPU's used in smart card applications can be read (both code i.e. algorithm, and data i.e. keys), sometimes with

- simple techniques (using nonstandard programming voltages to clear code protection fuses or misusing the manufacturers undocumented internal testing features), or with
- specialized, but still relatively cheap techniques (magnetic scanning of currents throughout the working chip) or eventually with
- expensive, but comprehensive techniques (acid washing the chip away layer at a time).

All of these have been done, and not always on very large budgets (e.g. just $10k could get you pretty far).

If a single smart card technology is used *widely* enough, the incentive of breaking it can become very high. For example, if a European Union -wide electronic cash card would be introduced, the incentives for criminals and organized crime to break it would be very large indeed.

> Even with the so-called secure smart card technologies, you can't expect total physical security for your code (algorithm) or any data (keys) on the card.

This is especially bad news for systems based on symmetric cryptography techniques, as the single key used both for encryption and decryption could be recovered from any of the many cards in circulation. This is why smart card applications need public key or zero-knowledge protocols and solutions.

This is also a problem for the Clipper chip proposal from NSA. If the Clipper would become mandatory and other crypto would be banned (as planned), the incentive for breaking it would also become extremely high, and somebody would surely pop the Clipper chip system open (however much it would cost) and find out the supposedly secret algorithm and any embedded keys for analysis.

5.4 Embedded Applications Need (Some) Security Too

Many embedded applications could use some level of real security. For example, remotely controlled car door locks, garage doors, TV remotes, remotely controlled model airplanes, could be made more secure by using real cryptographic security techniques in them.

Most of the time these applications are also extremely cost limited: They use the cheapest, $1-$2 price scale embedded controllers, which don't have much memory or computing power.

We are not aiming to build a totally 100% secure system here. We should keep in mind the trade-off between cryptography, resources and the required level of security. In many embedded solutions it may be enough to provide a system safe enough, for the particular application, and given the attack scenarios.

For applications where you need total security (e.g., digital cash), the protocols you need are quite heavy and would require high-end embedded controllers or smart cards, and very complex public key or zero-knowledge protocols, which are outside the scope of this paper.

Smart Card and Embedded Computing

Let's look at one popular and cheap embedded controller family, Microchip's PIC series, to find out what these environments look like. This family of embedded controllers is quite high in computing power (in comparison to other little embedded controllers) so they might make for a relatively good environment in cryptographic applications.

The computing power of these RISC pipelined controllers is around a maximum of 10 20 MIPS (instructions working on 8-bit data).

The working memory resources of chips in this family are from 24 to 256 bytes of RAM. Some models come with 64 bytes of EEPROM, which could be suitable for key and secret storage. The program memory capacity ranges from 512 instructions to 4096 instructions.

A typical widely-used model, the PIC 16C84, has a program memory of 1024 instructions, 36 bytes or scratchpad RAM, 64 bytes of EEPROM (nonvolatile) memory and a maximum performance of about 3 MIPS. This is our example target environment.

It's noteworthy to say here that the 16C84 chip has been found vulnerable. By using non-standard programming voltages, you can trick the chip into a mode that will allow you to clear the memory protection bit without clearing out all of the memory simultaneously (as would happen if it worked to its specification). As the 16C84 is a popular chip used in many applications, the vulnerability is rather serious, from the security viewpoint, as well as from the code copyright viewpoint.

Many other manufacturers also have comparable embedded controllers. These controllers sell a lot in the world. They are the kind of thing you can find in your computer mouse or microwave oven and other such little devices all around us.

6. A Minimalistic Zero-knowledge Protocol

6.1 Principles

In this paper we analyze a protocol with some degree of security and hopefully minimalistic resource use. We are willing to sacrifice some security for the simplicity of implementation and light resource use.

In a cryptosystem, the level of security required is a balancing act between apparent threats. We only need to make the breaking of the system not worth the perpetrators time and resources. For "boring" embedded applications, this security level is not necessarily very high.

We shall analyze a protocol where

- use a very light round zero-knowledge -based protocol (so that not much computing power is required at any given time), where we
- require each round to complete within a very short time limit (so that an outsider even with superior computing power doesn't have time to break the algorithm by brute force), and
- make sure that a quick dictionary-based attack (i.e. precalculated tables) attack is not feasible for an outsider with superior memory resources.

If the iteration round is light enough, we can possibly iterate many times to establish the credentials for the connection. If the embedded controllers used in the system are pretty fast, the resources required for an outsider to break the protocol by brute force would become be high enough to prevent attacks.

The approach of limiting allowable response time is also used in some existing protocols to avoid man-in-the-middle attacks (e.g. the Chess grand master problem or the Mafia fraud).

6.2 Implementation

As the chosen example target environment only has about 36 bytes of RAM available for the inner working of the protocol, and some RAM space must be reserved for other use, it seems that the key lengths and numbers used in the protocols must be within 32 to 64 bits (4 to 8 bytes) in length.

A 64-bit system should be strong enough to thwart most brute force attacks (e.g. DES keys are 56 bits). A 32-bit system is already suspectible to attack, as current desktop computers can have gigabytes of memory space to use. 64-bit key-space should be pretty good.

Every phase of the protocol must complete within a given time limit, just enough for the legitimate secret holder to calculate the reply, and transmit it. It is hoped that this time restriction will make man-in-the-middle and brute force attacks unfeasible.

A superior computing power intruder might realistically have 3 to 5 orders of magnitude more computing power than the legitimate system. Even if the brute-force calculation is fast, you will need to be able to break the problem within the given time limit, always. Making sure a brute force attack is unfeasible, it should be enough to make a brute-force attack take 10 or so orders of magnitude more computing power, i.e. a few bits worth of magnitude.

Victor must provide some random input for Peggy's problem, so that an impostor is not able to precalculate some set of problems and both solutions using his superior computing and memory resources.

6.3 Security Analysis

Memory (precalculation) attack

It is possible that an adversary with superior memory capacity could be able to use e.g. precalculated tables to speed up his attack, to be able to respond within the time limit.

As Peggy's problem is changed by the number given to her by Victor, she can't pre-build a set of problems and solutions to them.

Calculation (brute force) Attack

It is possible, that an enemy with superior computing power (i.e. who carries his all-powerful laptop computer to play the role of the little smart card) might be able to solve the problem in time to respond to Victor in time with the right, requested solution.

The time limits set in the protocol should make this attack unfeasible.

Post-mortem Brute-force Attack

An enemy or malicious Victor could save the conversation, and try to break it, later, off-line, without time, computation or memory limitations. This way, he could be able to find out the secret and then masquerade as the original Peggy (as he now would know Peggy's secret).

This attack is possible on other protocols too, but because this protocol uses a relatively weak single round, the possibility of this attack succeeding might make it a real danger to the security of this protocol.

One economical solution might be to look at the applications of this protocol: Non-critical embedded systems, each with their own secrets. It may not be economical for someone to break the system, if the possible benefits are very small.

Breaking a 64-bit secret is still relatively formidable effort by brute force, so in the environment where this protocol would be used, it should be enough.

Note that some of the numbers used in zero-knowledge need to have some special features (e.g., being prime), the brute force attack on this system may be much easier than the brute force attack on a conventional symmetric cryptosystem with 64-bits of real random key.

For example, if you know that breaking the secrets would take several days on a very powerful computer (you could even try it yourself in advance on your own system), you could, anyway, use these protocols in a one-day event for access control smart cards, with almost total confidence. When the adversary breaks the code, the particular secret(s) is (are) not in use any more.

6.4 Conclusions

The post-mortem calculation attack is a real risk with any weak or weakened cryptosystem. If the keys or secrets are not changed often, a malicious Victor or Peggy, or third-party eavesdropper may be able to find the secret by doing a brute-force attack on the recorded conversations. Especially in the case of minimalistic solutions, where the key length has been minimized because of real-world hardware limitations, it becomes a serious problem.

By balancing the security needs, applications and system capabilities it may still be possible to build systems with relatively good security, instead of no security as in most current systems.

It is evident from this study, that implementing real security does have quite large computational and memory

requirements. So, in applications where good security is necessary, high-powered embedded controllers should be selected, so that they can work with the full-strength cryptographic protocols.

Another possible solution would be to use symmetric-key cryptography protocols (which have relatively small computational and memory requirements), with designed-in features for the eventual loss of key secrecy through reverse engineering.

Joining theoretically good cryptographic techniques and protocols with real world limitations of small systems is sometimes extremely hard. Most studies are based on the availability of enough computing power and memory.

When you try to apply these techniques in very small systems, you will have to make compromises. Knowing the strength of your protocols, keys and the hardware and being able to balance and apply the system to your actual needs will be even more important than in traditional cryptographic applications.

References

[1]
David Chaum. *Security without identification*: Card computers to make big brother obsolete

[2]
Ivan Bjerre Damgård. *Zero-knowledge Protocols*, Århus University and CRYPTOMATHIC A/S.

[3]
Bruce Schneier. *Applied Cryptography*, Wiley & Sons, 1994, ISBN 0-471-59756-2, 1994.

[4]
Quisquater & Guillou. *Teaching Zero-knowledge protocols for your children*

[5]
DigiCash, Inc. Zero Knowledge Interactive Proofs

[6]
DigiCash publications. a list of publications by D. Chaum

[7]
Tatu Ylönen. crypto page

[8]
Has DSS Been Hacked?

[9]
UCL Crypto Group.
UCL Crypto Group home page

[10]
RSA Data Security, Inc. FAQ About Today's Cryptography

[11]
Gilles Brassard. A Bibliography of Quantum Cryptography

[12]
Erik Warren Selberg. How to Stop a Cheater: Secret Sharing with Dishonest Participation

[13]
Hadmut Danisch. The Exponential Security System TESS: An Identity-Based Cryptographic Protocol for Authenticated Key-Exchange.

[14]
Assorted Cypherpunks Topics

[15]
IBM Research. Digital Signatures Secure in the Strongest Sense Known

[16]
Security Papers

[17]

Rumor(tm) Database - cypherpunks