



Answer, ETIK, ELTE
Jedlik Ányos Program
RFID research project

Cryptographic Techniques for Physical Security (Seminar Notes)

Author: Daniel A. Nagy

ELTECRYPT Research Group

Supported by the National Office for Research and Technology (NKTH) in the
framework of Jedlik Ányos Program with number of tender
NKFP2-00027/2005.



Cryptographic Techniques for Physical Security (Seminar Notes)

Daniel A. Nagy

nagydani@epointssystem.org

January 30, 2007

Preface

The goal of this course is to familiarize participating students with the most important cryptographic techniques and design principles in the context of RFID (Radio Frequency IDentification) security. Unlike most courses and texts on cryptography, our primary focus is *not* the most traditional application of cryptography — enciphering of (possibly very large) textual messages.

Instead, we will study in detail several aspects of authentication, identification and authorization. In such applications, we will be typically dealing with short messages in low quantities. Instead of the usual long-distance point-to-point communication channel model, we will often assume short-distance broadcasts. Another important distinguishing feature of the security systems we will be studying is the fact that the computational resources of certain elements are severely constrained. Nevertheless, most principles and techniques studied in this course have applications far beyond RFID and are generally applicable in security system design.

The language of course notes is English, as most of the literature on cryptography and information security is available in this language and it is important that students of this subject become familiar with English terminology.

1 Introduction to Security

“Many counting rods is victorious over few counting rods,
How much more so over no counting rods.”
— Sun Tzu: Ping Fa (“The Art of War”, Denma Translation)

1.1 What is Security Engineering?

Security considerations always arise in the context of conflicting interests. Specifically, when there is a possibility (or *threat*) of such an action (called *attack*) that is beneficial for the initiating party, the *attacker*, but harmful to some other party, the *victim*. While reasoning about security, we often assume the point of view of the potential victim, referring to potential attackers as *adversaries*. Yet, it is very important to keep in mind that the interests of providers and consumers of security are not perfectly aligned either.

We are primarily concerned with *rational adversaries* (in the economic sense), who attack if they prefer the *gains* from the attack to the *cost* of performing the attack. While these are often difficult to quantify, using the usual micro-economic axioms about preferences, it is always possible to assign a scalar utility function to these gains and costs (see Debreu’s Lemma in the supplement). In practice, we usually express them in terms of monetary value. The matter is further complicated by the fact that we can only guess the preferences of the adversaries and attackers can only estimate their costs and the gains; perfect foreknowledge is rarely available.

Anything that imposes additional costs on the attacker is a *security measure* (a.k.a. *defensive measure*). These are not the only ways to mitigate threats (one can buy an insurance policy, for example), but alternatives are mostly outside of the scope of security engineering. Security measures fall in two main categories: *proactive* measures and *reactive* measures. Proactive measures make achieving the goals of the attacker more expensive, while reactive measures help punishing the attacker after completing the attack.

Security measures also have costs associated with them; security measures that cost more than the *loss* caused by successful attacks are not worth deploying. It is important to emphasize that security measures often affect more than just the costs of the attacker and the defender. As side effects, they can alter the gains of the attacker and the losses of the victim.

The losses of the victim and the gains of the attacker are often not commensurable, and the fact that we usually express them in the same (monetary or time) unit does not imply that they

are. The preferences of the two sides can be very different.

Security measures can successfully prevent attacks in three cases:

1. The adversary is not able to afford the increased cost of the attack.
2. The increased cost of the attack surpasses the gains (as perceived by the adversary).
3. The increased cost renders some alternative action preferable for the adversary.

Correspondingly, in two cases attacks defy countermeasures: when effective countermeasures are beyond the means of the victim (*predatory attack*), and when effective countermeasures are not worth taking (*parasitic attack*). A possible avenue of attack that would not be prevented by the security measures in place is called *vulnerability*.

Of course, the potential victim sleeps more soundly if the deployed security measures are of the first kind. For preventing the attack, these do not rely on the attacker being rational and well-informed, neither of which can be taken for granted. However, deploying such security measures is often not an option, either because they are unaffordable or because the potential losses do not justify the costs.

In many cases, we resort to security measures of the second kind. These are effective against rational attackers that know about the costs and the gains of the attack, but are not necessarily aware of alternatives. Performing the attack would cause net loss to the attacker, so it is reasonable to assume that it will not be undertaken.

Unfortunately, such security measures can also be unreasonably expensive, but even that is no reason to surrender. A security measure can make the attack expensive enough to make some alternative action on the part of the adversary (e.g. attacking someone else) preferable to attacking, even if the attack would still result in net gain.

Certain threats, however, are not worth defending against. In such cases, other means of damage control, such as insurance or backup solutions, should be considered.

Security systems are collections of security measures defending the same assets against a number of threats. Security engineering is the theory and practice of designing and maintaining security systems as to maximize the economic benefit emerging as a result of prevented losses from attacks and incurred costs of the deployed security measures. In the case of linear utility functions (which is often a reasonably accurate model assumption), security system design aims to maximize the *difference* between prevented losses and incurred costs.

As security measures prevent attacks by imposing costs on the attacker (cryptographic measures, in particular, impose computational costs) in such a way that the adversaries decide against attacking, a large part of security engineering is *threat modeling*, in which we estimate the potential attackers' capabilities, resources (budget) and likely preferences. Threat modeling is the topic of our next lecture.

In the second part of this lecture, we will elaborate on the notion of security.

1.2 What is “Secure”?

Security systems or components for such systems are often touted as “secure” by their respective vendors. Such absolute claims come across as naïve at best and misleading at worst. Security systems and their components always address specific threats. But, as discussed in the previous part, the effectiveness of security measures is meaningful only in the context of protected assets, which is often not available when evaluating security systems and their components. Since one of the most efficient ways of reducing the cost of security is to spread the considerable cost of development and testing over a large number of deployed instances, there are immense benefits (discussed later in more detail) from using standard components in security design. Thus, we need objective criteria for evaluating the security of components and systems, when no or little information is available about the protected assets and their value to the defending party.

Traditionally, in order to compare the security of various defensive measures, we use various metrics capturing the costs they impose on attackers (e.g. the minimal amount of time required to open a safe without knowing the combination, the expected number of trials for guessing a secret key, etc.). However, improvements in such metrics in components do not always translate into increased security for the system: Replacing a two-hour safe with a five-hour safe in a vault that is inspected every ten minutes is not likely to prevent attacks. Using encryption with a 256-bit symmetric key instead of 64 bits, will not prevent attacks either, if the key is derived from a 6-digit PIN.

As a relatively recent development, an approach borrowed from the economic analysis of efficiency of resource allocation has been adopted in security by Ian Grigg [3]. Let us call a change in a security system *Pareto-secure improvement*, if it may prevent attacks without allowing for other attacks. Within a given security system, a component the replacement of which *does not* result in a Pareto-secure improvement is called *Pareto-secure within the confines of the security system* in question. This can happen for two reasons: either because the cost of attacking this component is unaffordable for the adversaries (as in the example with the patrolled vault) or because attacking other components is clearly preferable (as in the example of the 6-digit PIN). A component that is a Pareto-secure choice for all (conceivable) security systems is called *Pareto-complete*.

From [3]: “Using a Pareto-complete component is always sufficient. Designers prefer and select Pareto-complete components in order to reduce local calculation costs, risks of incomplete calculations, external costs of verification, and risks of future changes weakening the system. If unavailable, a Pareto-secure component is also sufficient, but it carries risks that the security system may face unpredictable changes in the future that are dramatic enough to challenge the security. It also carries external costs of analysis, in that verifiers of security need to confirm that a component is indeed Pareto-secure within that system.”

For Pareto-secure systems, it is necessary that all components be Pareto-secure.

1.3 Mathematical Supplement

Debreu's Lemma: Let X denote a completely ordered subset of a finite-dimensional Euclidean space. If for every $x' \in X$ sets $\{x \in X | x \leq x'\}$ and $\{x \in X | x' \leq x\}$ are closed in X , then an ordering-preserving real-valued continuous function exists over the elements of X .

See [1] for more details.

1.4 Exercises

1. What security measures are taken in a car? What do they protect? What threats do they counter and how? Which ones are proactive, which ones are reactive?
2. One security measure in 2004 S-Class Mercedes is a fingerprint scanner preventing unauthorized persons from starting the engine. Analyze this security measure from a costs-benefits point of view (hint: this measure was removed from the 2005 model).
3. Analyze the security of the public transit system against freeriders.
4. What components in your home security system are Pareto-secure? Are there any Pareto-complete ones?
5. Nonces are random codes used to distinguish each authorization session; a security measure often used for preventing attacks by replaying messages recorded in the past. How many binary digits would be a Pareto-complete choice for the length of nonces?

2 Multiple Threats and Defenses

“And so it is said—

Know the other and know oneself,

Then victory is not in danger.”

— Sun Tzu: Ping Fa (“The Art of War”, Denma Translation)

2.1 Threat Modeling

From [2]: “Threat modeling is, for the most part, ad hoc. You think about the threats until you can’t think of any more, then you stop. And then you’re annoyed and surprised, when some attacker thinks of an attack you didn’t.”

All security systems rely on some implicit or explicit assumptions about the nature of the threats they face. In many cases, security systems fail not because their way of countering anticipated threats is inadequate, but because attackers challenge the very assumptions upon which the security of the security system depends. Getting the threat model right is paramount to designing successful security systems. However, starting the above process from scratch each time is prone to result in unnecessary errors and wasted effort. Instead, Schneier recommends a systematic way of looking at threats, which allows us to re-use results from past work and use our present results in the future: organizing threats into what he calls an *attack tree*.

Some action can be considered an attack either because it causes losses directly or because it makes other attacks possible. Thus, following [2], attacks can be organized into a tree (actually, an acyclic directed graph) according to what other attacks do they make possible and whether they are sufficient or necessary for performing those attacks. Equivalently, one can think about these trees as (monotonous) logical expressions on the leaf actions, where sufficient actions are joined by *or* clauses, necessary actions are joined by *and* clauses.

Attack trees reflect the (anticipated) options of attackers to achieve their goals. An actual attack consists of actions for which the logical expression corresponding to the attack tree is true, when setting the variables corresponding to these actions to “true” value. Because of the monotonous nature of the expression, any superset of actions is an attack as well. Conversely, if certain actions do not result in achieving the goal, neither does any subset thereof.

Such a representation allows for a lot more than just saving some work on enumerating threats; it allows for predicting attack preferences and the costs of attacks, it allows for in-

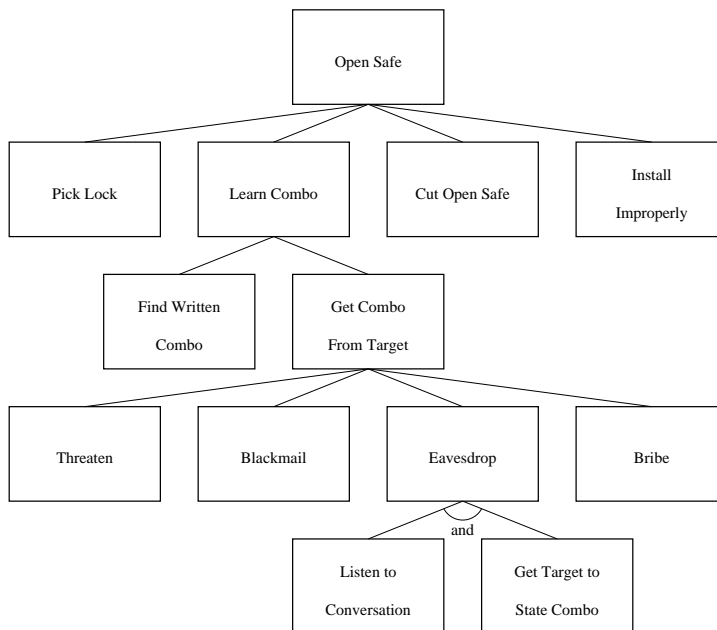


Figure 2.1: Attack tree (example from [2])

incorporating lessons learnt from past incidents that happened to different systems into security design, it allows for evaluating the effectiveness of various security measures and a lot more. Most importantly, it allows for efficient division of labor in security engineering.

Using the attack tree for assessing the (minimal) costs of attacks and predicting the likely choices of a rational attacker is quite straightforward. The minimal cost of performing an action corresponding to a specific node of the attack tree can be expressed as the minimum of the costs of sufficient children or the total cost of necessary children (in case of linear utility functions, the sum of the minimal costs thereof).

Even without knowing the preferences of our adversaries and the ability to model them with some scalar utility function, we can use the attack tree to determine which attacks are possible, given some assumptions about the capabilities of the adversary.

The evaluation of the effectiveness of various security measures is also significantly aided by representing threats in attack trees; security measures typically increase the cost of performing specific actions. How this changes the overall costs of attacks can be deduced from re-evaluating the attack tree.

2.2 Cooperation in Security

Several players with slightly different roles participate in the security process. They typically fall in one of the following categories:

subject: one who would suffer the losses caused by the attack. Typically a person or an organization, defining the security policy.

agent: one acting on behalf of the subject implementing the security policy, though not necessarily affected directly by the attacks. An agent may suffer only part of the losses caused by successful attacks. Typically an employee, a contractor or a customer.

provider: one whose sole responsibility towards the subject is designing and/or deploying security solutions. Unlike agents, providers do not necessarily suffer any direct losses from attacks; all losses of the provider may be of indirect nature, through liabilities and reputation.

trusted party: from [2] (quote attributed to “a friend at the NSA”): “Someone whom you know can violate your security policy without getting caught.” Trusted parties are typically independent organizations (or individuals) involved with many other security systems of similar kind.

While they have a common interest in security, their motivation and priorities may be very different. Of course, these roles are not necessarily distinct. When the subject is a group, the members are necessarily agents, who share some but not necessarily all interests of the group. When a security solution is developed, deployed and operated in-house (a scenario worth considering only for the extremely well-funded), providers are agents as well.

All other participants can violate the security policy of the subject and thus have to be included in the threat model on one hand and provided with the necessary incentives on the other hand. Trusted parties (often called *trusted third parties*, when used to help transactions between parties with slightly different interests) deserve special attention, as they must, by definition, be distinct from the subject and thus necessarily constitute a weakness in any security system. Szabo eloquently argues in [4] the reasons (and some methods) to avoid TTPs when possible.

Combining a trusted party with any other role (except the provider) will necessarily result in a conflict of interests compromising the integrity of one or both of the roles. Also, it is a fallacy to assume that the costs of the trusted party can be arbitrarily reduced by spreading the cost among the many users of the trusted party, because the more users rely on the trusted party the more is there to be gained from attacking it, thus the more costly its security becomes. Actually, common trusted parties can easily become an obstacle for scalability for this very reason.

2.3 Secrets

Secrets (deliberately withheld information, known to some parties and unknown to others) play an important role in security in general and are central to cryptographic applications. Secrets can be protected assets as well as means to protect other assets. In this section, we will discuss some basic aspects of dealing with secrets.

2.3.1 Shared Secrets

Unfortunately, two very different concepts are called *shared secret* in English. One is the same secret known to more than one party, the other is a secret that can be assembled from different secrets possessed by different parties. In this section, we deal with the former.

Shared secrets have the following undesirable property: the losses incurred by the leaked secret are suffered by all parties, while the costs of protecting the secret are born by each one individually. Moreover, the gains from attacking the system by leaking the secret are won only by the *traitor*. Thus, depending on the actual figures for losses, costs and gains, there is a potential for what is known in game theory as *prisoner's dilemma* or *tragedy of commons* (see the supplement).

As the aphorism attributed to Benjamin Franklin goes, “three can keep a secret, if two of them are dead.” There is a qualitative difference between secrets shared by two parties and secrets shared by three or more parties. The difference is the following: if the secret is leaked, in the case of two parties, it is known (to them) who the source of the leak was, so reactive security measures imposing various costs on the traitor are possible. If a secret is known to three or more parties, there is no way to defend it in the (likely) case of the tragedy of commons. Thus, secrets shared by three or more parties are to be avoided in security systems whenever possible.

Of course, shared secrets between two parties are also more vulnerable than secrets known to one party, so it is instrumental to keep the value (both the loss of the victim and the gain of the attacker) of the secret bounded from above. Secrets the value of which keeps increasing over time without bounds should not be shared.

Public key cryptography is usually the tool of choice to avoid shared secrets, because it allows the use of a secret in one party's possession by others without the need to share it.

2.3.2 Global Secrets

A secret upon which the security of a whole system depends (e.g. a master key to all locks or a root password to all computers) is of equal value to the security of the whole system. During the lifetime of the system, its value is bound to increase. Thus, the protection of this secret easily becomes the most expensive part of the security system. If the secret cannot be changed (that

is the old secret cannot be rendered worthless for attackers), certain security measures taken in the past to protect it (the choice of the type of the key, etc.) may become inadequate over time. Using a probabilistic model, if the probability of a compromise during a fixed period of time has some lower bound (determined by decisions taken in the past), the probability that the compromise has not occurred converges to zero exponentially fast. Thus, parameters that cannot be changed (e.g. the structure of the security system, decryption keys for information that is available in encrypted form, or biometric information of authorized personnel) *must not* be used as global secrets. If possible, global secrets must be avoided in general.

A global secret that is shared (especially between three or more parties) is a recipe for disaster. Such mistakes in security design have historically lead to spectacular failures.

2.3.3 Using Secrets as Security Measures

When the cost imposed by a security measure on the attacker is that of finding out a secret, very small secrets are sufficient to thwart attacks by *brute force* (enumeration of all possibilities), since the size of the secret can be as small as the logarithm of the number of possibilities. Such secrets that are parameters of the security system set by the subject or its agents are called *keys*.

The workings of the security system are known to the provider. If the security of the system or its components depends on the secrecy of information other than the keys, it becomes a shared secret between the subject and the provider, unless they are the same. If the same system or component is provided to many subjects, we have a secret known by many parties, which is extremely difficult to keep. This has two very important implications:

For consumers shopping around for security systems or components, it means that providers' claims that the security of the system would be weakened by disclosing its inner workings are best believed and such systems and components (and providers) are best avoided. For providers, in turn, it means that publishing the details of what they are selling makes good marketing sense; if their solutions are secure even under full disclosure of all details (except the keys, of course) it inspires confidence.

All this, however, does not imply that subjects should also disclose everything. From their point of view, there is often no benefit from disclosing the details of the security system they use, while it may add to the costs of the attacker to keep certain things secret or even lie about them. It is very often beneficial to deceive the attacker regarding the costs and benefits of the attack; if the bluff is expensive to call and cheap to make, it is a valid security decision. The extreme case is the so-called *dog-sign security*, when the dominant cost on the attacker is that of finding out what the costs of an attack would be (like placing a "beware of savage dog" sign on a fence around a house with no dog). When the subject and the provider are the same, very sophisticated dog-sign security measures become perfectly rational choices (e.g. there are good reasons to believe that the US ballistic missile defense system is a dog-sign security measure).

2.4 Mathematical Supplement

Shared secret payment matrix: One holder of the shared secret can choose to protect it, and pay the costs (denoted by C) or to divulge it and not to pay the costs (which include the opportunity costs from receiving the benefits going to the traitor). If someone fails to protect the secret and it leaks, the losses that occur to the player in question are denoted by L . In the case of two parties, there might be some penalty P that is paid by the traitor, but not the others. Thus, we get the following payment matrix for one holder of the secret, given the two possible strategies (protecting vs. divulging) and the two possible behaviors of the rest (with non-negative C , L and P):

$$\begin{pmatrix} -C & -L - C \\ -L - P & -L - P \end{pmatrix}$$

The benefits from keeping the secret, if everybody else does equal $L + P - C$, if someone else also divulges the secret, it becomes $P - C$. If $C > L + P$, the rational behavior is to become a traitor, no matter what others do. Keep in mind that $P = 0$ for more than two participants, L is the loss to the individual (not the group) and C includes the opportunity cost of not receiving the bribe, so the dangerous situation is likely to be the case.

2.5 Exercises

1. The assumption that the success of the attack is a monotonic function of the various actions is not self-evident. Some avenues of attack may be mutually exclusive. Think of scenarios when the proposed threat model could lead to misplaced security measures.
2. The popular on-line auction web-site, eBay, combines the roles of a trusted third party (making sure that the auction is fair, in exchange for a percentage of the winning bid) and an agent (by automatically bidding on behalf of the users, who only need to tell their highest bid). This necessarily allows eBay to attack the users and get away with it. How?
3. The default configuration of ssh, a remote administration tool, disables remote root logins. Why?
4. Analyze the economics of disclosing vulnerabilities in widely deployed security systems. Find the optimal strategies for various players. Verify your conclusions by observations, if possible.

3 Cryptographic Primitives

“Tastes do not exceed five,
Yet all their variations cannot be tasted.”
— Sun Tzu: Ping Fa (“The Art of War”, Denma Translation)

3.1 Introduction

Cryptographic security measures, by their very nature, are much cheaper to deploy than to develop and test. Thus, in cryptography, development and testing costs can be spread over a large user base, thus reducing the costs of each individual. Also, because of the shaky mathematical foundations of cryptography and the inherent difficulty of the task, the testing of cryptographic components can never be considered finished. Popular components that are used in defending many assets (and thus would result in immense gains for successful attackers) inspire confidence (if they wouldn’t be secure, they must have been attacked by now). Yet, there is no guarantee that successful attacks will not be devised against any specific component.

Thus, “following the herd” by using popular standard components in security design is a wise strategy for several reasons: first, such components are subject to intensive research, both well-intended and malicious, so it is highly likely that flaws will be revealed *before* our particular security system gets attacked, secondly, in the event of discovering a weakness in such a component, the security community is highly motivated to develop alternatives. In most cases, alternatives are developed long before the need to replace a component becomes pressing.

In order to facilitate the use of standard components, it is advantageous to design cryptographic systems and protocols using abstract components with well-defined security objectives (called *primitives*) and pick actual implementations afterwards. In this lecture, we will introduce some of the most popular cryptographic primitives and discuss their use in security systems.

This chapter covers several lectures.

3.2 One-Way Functions

3.2.1 Introduction

Mappings for which finding a pre-image (an argument yielding a specific function value) is computationally difficult are called one-way functions. The overwhelming majority of cryptographic measures hinges on the security of one-way functions. It is important to emphasize that the notion of “computationally difficult” does not have a generally applicable mathematical definition. The usual definitions from complexity theory, while useful, are sometimes neither necessary nor sufficient for security. Furthermore, because of open problems in complexity theory, it is not clear that families of functions, where the evaluation the function requires polynomial resources (in terms of the length of the argument, in the worst case), while finding a pre-image requires exponential resources (in the typical case) exist at all. Also, one must exercise caution when applying such asymptotic criteria to finite problems.

Depending on the size of the argument and the size of the function value, useful one-way functions fall into one of the following categories:

Substitution function: The argument and the value are of fixed, equal length.

Compression function: The argument and the value are of fixed length, with the argument being longer.

Expansion function: The argument and the value are of fixed length, with the argument being shorter.

Hash function: The argument is a sequence of arbitrary length, the value is of fixed length.

Keystream generator: The argument is of fixed length, the value is a pseudorandom sequence that can be evaluated to an arbitrary length.

In addition to the one-way property, we often require *collision resistance*, which means that finding two different arguments for which the function yields the same value is also difficult (or even impossible). For compression functions and hash functions (a.k.a. *message digests* or *digest functions*), collision resistance typically means that finding two colliding arguments requires, on average, resources proportional to the square root of the range of the function.

3.2.2 The Random Oracle

For design and evaluation purposes, we often use the *random oracle* [5] as a common abstraction for one-way functions mentioned above. The random oracle, which is available to all parties (including the adversaries) does the following: when given a binary vector of arbitrary length,

she responds with a sequence of independent random bits coming from a Bernoulli distribution with parameter $p = \frac{1}{2}$ until stopped. The only constraint on the oracle is that given the same vector, she responds with the same sequence.

In security proofs, we often replace certain parts of the security system with queries of one or more random oracles (available to all parties, including attackers) and prove security for this *ideal* system. However, this method has received ample criticism on the grounds that it is possible to design protocols that are secure under random oracle assumptions, but *any* implementation would render them insecure (see [6] for details). Clearly, security under random oracle assumptions is necessary; the exact conditions when it is sufficient is still subject to intensive research. Random oracle assumptions are stronger than mere collision-resistance.

3.2.3 Linear Cryptanalysis

Supposedly one-way functions are said to have a *linear bias*, if the modulo 2 sum of some input bits and some output bits is 0 with a probability different from $\frac{1}{2}$ (where the probability measure is defined uniformly over all possible inputs of equal length).

Such bias can be fruitfully exploited for finding pre-images, collisions and other violations of security assumptions. Linear cryptanalysis was first presented by Matsui and Yamagishi in 1992 [7] and has drawn the attention of the cryptographic community when Matsui successfully applied it to the US government Data Encryption Standard (DES) cipher [8].

3.2.4 Differential Cryptanalysis

Differential cryptanalysis deals with *pairs* of inputs and corresponding outputs. If certain differences in inputs lead to certain differences in outputs with a probability that is larger than what is statistically warranted then the function is said to have *differential characteristics*.

Such characteristics can be used for statistical attacks in a similar manner. Although differential cryptanalysis was first presented by Biham and Shamir in 1990 [9], it has been discovered more than a decade earlier at the US National Security Agency (NSA) and possibly elsewhere.

3.2.5 Application Notes

One-way functions with no additional special properties have several direct applications. In this section, we give a brief overview of some of these applications.

3.2.5.1 Authentication Using a Publicly Readable Database

Suppose that authorized access is granted to parties presenting some secret pass-code over a channel that is secure against eavesdropping (identification is done by some other means).

The equipment performing the authorization is also assumed to be trustworthy. However, the database which is used to verify what codes are authorized is readable by potential attackers.

We want to impose computational costs on attackers who want to learn valid pass-codes from the database. An obvious thing to do is to store one-way mappings of passcodes using some one-way function. During authorization, the same function is applied to the presented pass-code and the result is compared to that in the database.

If, for some reason, there is a possibility that several identities share the same pass-code, using the above solution would tell adversaries which ones have identical pass-codes. In order to deny attackers this information and make sure that all records are different, a unique sequence (called *salt*) can be added to each pass-code before the one-way transformation. The salt value should be stored along with the one-way image of the pass-code so that it is available at the time of verification.

If the entropy of pass-codes is low enough so that an attack by enumeration (a.k.a. *dictionary attack*) may yield a valid pass-code in a reasonably short time, salting imposes the additional cost of having to do the enumeration for each identity separately. An additional measure may be using several iterations (typically tens of thousands) of the one-way function on the salted pass-code. This may increase the time required for each enumeration by a multiplicative factor rendering the exercise excessively expensive for the attacker at the cost of introducing additional delay at verification time. However, the effectiveness of this measure may be questionable: attackers may have access to a lot more computational power than that available in the equipment used for verification. Also, there is no guarantee that the iterated function cannot be computed more efficiently. Finally, it is worth mentioning that a common mistake is using an iterated one-way function without salting. This defeats the purpose of the security measure, as the dictionary attack may be performed rapidly using a dictionary of pre-computed images; the additional cost imposed on the attackers can be spread over an arbitrary number of attacks on similar systems.

3.2.5.2 Authentication Over an Insecure Channel

If all the equipment used in the authentication can be considered trustworthy (that is it is protected by various physical security measures, as is often the case in RFID), but the communication channel is easy to eavesdrop (such as a radio channel), the proving party may generate a unique salt and send the one-way image of the salted pass-code and the salt to the verifying party, which, in case of successful authentication records the salt value to prevent attacks by replaying recorded messages.

There are other similar authentication protocols (e.g. when the salt, called *challenge* in this context, is provided by the verifying party), which will be discussed in detail later.

Using simple one-way functions, it is not possible to authenticate over an insecure channel using a publicly readable database. For that, one needs either *commutative* or *trapdoor* families

of one-way functions, both of which will be discussed later.

3.2.5.3 Stream Cipher Using a Keystream Generator

Keystreams can be used as one-time pads. It is very important not to reuse the same keystream for encrypting two different streams. In order to use the same key for encrypting different streams, one can use salting techniques similar to those discussed in Section 3.2.5.1 and prepending the salt vector to the encrypted keystream. The actual argument for the keystream generator is either the concatenation of the key and the salt, or a (possibly iterated) one-way image thereof. The role of the second one-way function (typically a compression function or a hash function) in this case is to thwart related key attacks in case the key-stream generator turns out to be vulnerable to such. One alternative to iterated compression or hash functions is discarding a large part of the keystream before using the rest as a one-time pad.

3.2.5.4 Ciphertext Feedback (CFB) Encryption Using a Compression Function

Let $f : \{0, 1\}^m \rightarrow \{0, 1\}^n$ denote a one-way function where $m > n$. We can use this compression function to encrypt a sequence of plaintext blocks $P_0, P_1 \dots$ of n bits each into a sequence of ciphertext blocks of identical length $C_0, C_1 \dots$ using a key $K \in \{0, 1\}^{m-n}$ and a known *initialization vector* $I_0 \in \{0, 1\}^n$ that should be different for different messages encrypted with the same key.

Encryption goes as follows: for each $i \in \mathbb{N}$, $C_i = P_i \oplus f(I_i K)$ and $I_{i+1} = C_i$, where \oplus denotes bitwise modulo 2 sum (exclusive or). For decryption, we need the same f , K and I_0 : $P_i = C_i \oplus f(I_i K)$ and $I_{i+1} = C_i$.

In a more general setting, one can use the same compression function f to encrypt a sequence of blocks at most n bits each. Let us denote the block length by $k \leq n$. In this case, $C_i, P_i \in \{0, 1\}^k$ and $I_i \in \{0, 1\}^n$. Instead of $f(I_i K)$, only the first k bits should be used and $I_{i+1} = I_i^{(n-k)} C_i$ that is I_{i+1} is the concatenation of the last $n - k$ bits of I_i and C_i .

Unlike the stream cipher in Section 3.2.5.3, this method of encryption is self-synchronizing in the sense that a decryption using the correct key K will result in correctly decrypted blocks after a while, even if decryption was started in the wrong state.

It is also worth mentioning, that compression functions can be used as keystream generators as well, using either the so-called *output feedback* (OFB) or *counter* (CTR) mode. We will revisit this mode of operation when discussing implementation details of various cryptographic primitives.

3.3 Homomorphic One-Way Functions

3.3.1 Introduction

A one-way function f is called *homomorphic* with respect to some algebraic operation \odot defined on its domain if it is cheap to compute $f(X \odot Y)$ from $f(X)$ and $f(Y)$, without knowing the values of X or Y . For example, it is conjectured that in certain finite groups exponentiation is a one-way transformation (that is g^x is a one-way function of x , where g is a generator element of the group in question); it is homomorphic with respect to addition modulo the order of g , since $g^{x+y} = g^x g^y$.

3.3.2 Application Notes

In some applications, such homomorphism of one-way functions can be exploited by attackers, just like any other algebraic property, as it may yield non-trivial information about the argument of the one-way function, given the value, which is undesirable in general.

However, homomorphic one-way functions (perhaps, with other special properties) can be used for securing distributed computing (so that those performing parts of some computational task cannot learn sensitive information about the whole). Typical uses include voting, payment and other applications in financial cryptography.

Homomorphic one-way functions are important building blocks of more sophisticated cryptographic primitives, such as blind signatures.

3.4 Block Ciphers

3.4.1 Introduction

Block ciphers are pairs of special compression functions $e : \{0, 1\}^{n+k} \rightarrow \{0, 1\}^n$ and $d : \{0, 1\}^{n+k} \rightarrow \{0, 1\}^n$ such that $d(e(XK)K) = X$ and $e(d(XK)K) = X$ for any $X \in \{0, 1\}^n$ and $K \in \{0, 1\}^k$. Thus, block ciphers can be used to encrypt and decrypt blocks of n bits using a symmetric key of k bits (n and k are called, respectively, *block size* and *key size* of the block cipher). To reflect this use, we often denote $e(XK)$ by $e_K(X)$ and, similarly, $d(XK)$ by $d_K(X)$ and regard a block cipher as a family of automorphisms.

The algebraic properties of block ciphers (regarded as automorphisms) determine how they can be used. For general-purpose block ciphers, it is desirable that they do not have any algebraic structure; since there are $2^n!$ possible automorphisms, a block cipher with similar block and key lengths (that implies $2^k \ll 2^n!$) can easily avoid being closed under composition.

3.4.2 Application Notes

Either direction of the block cipher can be used as a general-purpose compression function, but block ciphers are typically slower than compression functions without pairs. For example, PGP-compatible encryption systems use only the encryption direction of the supported block ciphers as a compression function for CFB encryption; the decryption direction is not used.

Block ciphers can be used in a number of other ways to encrypt and decrypt streams of data (e.g. ECB, CBC, etc.), which are outside of the scope of this course. For our purposes, block ciphers will be used to encrypt and decrypt blocks of data. It is generally safe to encrypt different blocks using the same key, especially if the number of blocks is insufficient for statistical attacks such as those mentioned before.

3.5 Commutative Block Ciphers

3.5.1 Introduction

A block cipher for which $e_A(e_B(X)) = e_B(e_A(X))$ and $e_A(d_B(X)) = d_B(e_A(X))$ is called *commutative*. For a long time, commutative block ciphers were not even considered, as they were deemed useless. In an undergraduate paper, Ralph Merkle proposed to use commutative ciphers to achieve something that was thought to be impossible for millennia: to transmit a message over a communication channel that can be eavesdropped, without a pre-shared secret. In his paper, he notes that using any commutative cipher (such as one-time pads) won't work; one needs commutative block ciphers (although he didn't call them such). The paper was rejected on the grounds that commutative block ciphers could not exist, as they would allow for something clearly impossible.

Thus, public key cryptography was not born until the seminal paper by Diffie and Hellman in 1976 [11]. The contribution of Merkle was acknowledged in the follow-up patent [12] granted in 1980. Several government agencies (especially GCHQ in the UK) claim to have invented public key cryptosystems before that, but there is little evidence of actually using it.

3.5.2 Application Notes

Merkle proposed to use the following three-way communication for sending a secret message M from Alice to Bob:

1. Alice generates a secret key A and sends $e_A(M)$ to Bob.
2. Bob receives the message, generates a secret key B and sends $e_B(e_A(M))$ back to Alice.

3. Alice receives this message and removes her encryption by applying d_A to it: $d_A(e_B(e_A(M))) = e_B(M)$. She sends $e_B(M)$ back to Bob.
4. Bob decrypts the message by applying d_B : $d_B(e_B(M)) = M$.

If the goal is merely to share a secret (which can be later used, for example, to encrypt communication by using it as a symmetric key), one can use what is known as Diffie-Hellman *key agreement* (or *key exchange*) protocol:

1. Alice generates a secret key A and sends $[G, e_A(G)]$ to Bob, where G is some public element in the domain of the block cipher called *generator*.
2. Bob generates a secret key B and sends $[e_B(G)]$ back to Alice.
3. At this point, both of them can calculate $e_A(e_B(G)) = e_B(e_A(G))$, which is their shared secret.

Note that in general the so-called *Diffie-Hellman problem*, which is devising $e_A(e_B(G))$ from $e_A(G)$, $e_B(G)$ and G , is *not* equivalent to reversing e , which is only *sufficient* for solving it. The actual complexity of the D-H problem is subject of intensive research for different commutative block ciphers. In cryptographic proofs, we often use the *Diffie-Hellman* assumption, which is assuming that the D-H problem is hard for our choice of commutative block ciphers.

Commutative block ciphers can be used to implement other cryptographic primitives as well and are among the most popular tools in public key cryptography.

3.6 Zero-Knowledge Proofs (ZKP)

3.6.1 Introduction

This cryptographic primitive lets Alice prove the possession of a secret to Bob without either revealing it to Bob or allowing Bob to prove Alice's possession of that same secret to a third party; an eavesdropper of their communication can learn nothing.

The protocol (which can be iterated, if necessary) works as follows:

1. Alice *commits* to her secret by revealing a (special) one-way function thereof.
2. Bob *challenges* Alice to calculate a function of the secret to which she committed and the challenge, which he is able to calculate from the commitment and something else he knows (but Alice does not).
3. Alice *responds*, by sending the result of the calculation to Bob.

This information is already known by Bob, so he cannot use it to prove anything to a third party.

The direct application of this primitive is authentication without leaving an audit trail. It can also be used to implement other cryptographic primitives.

3.7 Message Authentication Codes (MAC)

3.7.1 Introduction

Message authentication codes are “keyed hash functions”: one-way functions $m : \{0, 1\}^k \times \{0, 1\}^* \rightarrow \{0, 1\}^n$ where k is the key size and n is the size of the MAC. In plain English, the MAC is calculated from a k bit *key* and a message of arbitrary (finite) length. In some, but not all applications, collision resistance is a requirement, so Pareto-complete implementations should be collision resistant in addition to the usual one-way requirement.

It is cheap to compute the MAC of a given finite-length string using a known key, but calculating a valid key given the corresponding key and MAC, as well as finding a string that produces a given MAC with a given key is (prohibitively) expensive. Furthermore, it is expensive to forge a MAC of a given message without knowing a key, with which a large number of messages (even if those are adaptively chosen, but not equal to the target) with corresponding MACs are available to the attacker.

The strongest security requirement with respect to some MAC function is that of security against *existential forgery*, which is, correspondingly, the weakest adversarial goal. Successful existential forgery entails the creation of a valid message–MAC pair with no constraints on the content or the length of the message by an attacker that does not possess the secret key. A MAC construction that is secure against existential forgery is also secure against all other attacks, as all successful attacks against a MAC function allow for existential forgery.

In the context of information available to the attacker, the weakest adversarial goal is that of *existential forgery with adaptively chosen plaintext*; in this case, the attacker can ask the holder of the MAC key to generate valid MACs for arbitrary messages and even make the plaintext of these messages dependent on the result of previous requests. The attack is successful, if the attacker can present a message with a valid MAC that the holder of the MAC key *has not* computed. For a Pareto-complete MAC function, even this attack is prohibitively expensive for all possible attackers with reasonable computational constraints.

The strongest adversarial goal against a MAC function short of complete break (retrieval of the secret key) is that of *universal forgery*, when the attacker successfully computes the MAC for *any* message. In between, there is a wide range of *selective forgery* attacks, when the attacker can forge MACs on certain kinds of messages but not others.

Finally, MAC functions need to have the same properties as one-way hash functions. In fact,

a MAC function with a publicly available key *is* a secure hash function.

3.7.2 Application Notes

MACs are typically used as an integrity protection measure for plaintext information handed over to some untrustworthy party for recording. MAC functions allow for storing a single secret key (with adequate protection) by the verifier and handing out large numbers of integrity-sensitive messages to untrustworthy parties. Alternative solutions include storing the hash value (or even the complete plaintext) of each message, which is obviously more expensive. Thus, MACs are popular components in authentication and authorization tokens.

Depending on whether the integrity-sensitive messages are generated by some untrustworthy source or the holder of the MAC key, collision resistance, respectively, may or may not be required of the MAC function.

In the design and evaluation of security systems relying on MACs, it is common to treat the MAC function as a random oracle with the message plaintext and the MAC key on its input. If the used MAC function is secure against existential forgery with adaptively chosen plaintext, this random oracle assumption may be justified in formal proofs of security.

3.8 Digital Signatures

3.8.1 Introduction

As a concept, the digital signature was first introduced in [11] in 1976. The first construction for obtaining digital signatures was published two years later, in [13]. In short, it is the public key variant of MAC, where the integrity of the message can be verified using a public key. It is important to note that before the above mentioned publications, the two terms (“digital signature” and “MAC”) had been used synonymously in cryptographic literature. It still happens occasionally, especially in texts aimed at non-specialists.

The most important property of digital signatures in addition to those of MACs is *non-repudiation*, which means that a digital signature constitutes strong evidence that it has been calculated using the private key corresponding to the public key with which it verifies correctly, thus authenticating both the signed document and the signer. Much has been written about the essential differences between this technical use of non-repudiation and the legal term (see [16] for a detailed explanation). Yet, this — perhaps unfortunate — coincidence in legal and technical terminology is still causing widespread confusion, which is reflected in some particularly unhelpful legislation in many jurisdictions.

3.8.2 Application Notes

From a security engineering point of view, a digital signature is a mere authenticity and integrity protection measure, which is verifiable by parties that cannot be trusted with providing authentication.

Digital signatures can be used for authenticating both communication and stored data. Thus, they are particularly useful for providing authentic transcripts of communication sessions or parts thereof.

3.9 Exercises

1. Assume that the best way of breaking the key of some block cipher e_K, d_K is enumerating all possibilities, until the key is found. The composition of two such block ciphers $e_{K_1}e_{K_2}, d_{K_2}d_{K_1}$ is a block cipher with double key size. Can the key of this block cipher be retrieved faster than enumerating all possibilities for both halves of the key, assuming that the block cipher has no algebraic structure?
2. Construct a protocol for secure authentication over a channel open for eavesdropping with a publicly readable database using commutative block ciphers.
3. Implement ZKP using the other primitives.

4 Randomness

“He changes his camp,
Makes his route circuitous,
Preventing the people from obtaining his plans.”
— Sun Tzu: Ping Fa (“The Art of War”, Denma Translation)

4.1 Introduction

The notion of randomness in security engineering is quite different from that in other sciences (a fact unfortunately often ignored with perilous consequences). In security, the randomness of a value means high costs for the adversary to learn or guess anything about it, no more and no less. The higher the cost, the more random the value. This immediately implies that randomness is highly context dependent; it depends on who the adversaries are, what information is available to them (and what the costs of learning or guessing other information are). In cryptography, we typically use random bits. In this lecture, we shall discuss where to get them and how for various purposes.

In order to be able to use the mathematical tools of probability and statistics, we need to define a probability measure on the different values of random bits (and other observables). To this end, we evoke the notion of a rational attacker maximizing his expected net gains. Consider the actual strategy of the rational attacker and a set of distribution-dependent optimal strategies including this one. The distribution corresponding to the actual strategy is used. Note that this is not necessarily unique. Note, furthermore, that it is often not known in advance with certainty what the rational attack strategy is. In such cases (i.e. in typical practical ones), the probability distribution assigned to the values of random bits depends on our assumptions about the attacker.

Strictly speaking, statistical properties such as independence from observables, (jointly) uniform distribution and its consequences, are neither necessary nor sufficient for security, albeit highly desirable in most cases. Random bits with a non-uniform joint distribution can be Pareto-secure in a given context. For Pareto-completeness, the above mentioned statistical properties are necessary in the majority of cases, because the cost of guessing the values of a number of random bits with a non-uniform joint distribution is lower than that of guessing the values of the same number of jointly uniformly distributed random bits.

4.2 Obtaining Random Bits

Since the notion of randomness is context dependent, so is the best way of obtaining random bits. Cryptography (giving the adversaries problems in excess of their computational capacity available for attack) can help us getting more and better random bits out of a few poor ones by feeding them into one-way functions. As long as the adversary does not know (or cannot guess) something about all the inputs of the one-way function, the output is random (according to random oracle assumptions about the one-way function).

In the simplest example, we have a secret K_0 from a large enough set that the adversary has no hope of guessing. Then, we can get any number of random bits by incrementing a counter i of appropriate size and feeding the secret and the counter into a one-way function H that provides at least one bit on the output.

$$R_i = H(K_0, i)$$

This is a fine method, if all the assumptions are correct. However, if there is a non-negligible chance that the attacker may learn the initial secret, while having a reasonably accurate estimate on the counter value, randomness vanishes.

But what if our one-way function is less than perfect? In the above example, the inputs to the one-way function are closely related. In fact, subsequent ones differ only in two bits on average. If the secret K_0 is not known to the attacker, neither is $K_1 = H'(K_0, i)$, where H' is another one-way function, different from H . Thus, the sequence

$$R'_i = H(K_i, i)$$

where

$$K_i = H'(K_{i-1}, i)$$

for $i > 0$ is as random as R_i , but less vulnerable to related-input attacks against the used hash function (in practice, H and H' are two halves of the same hash function). R' , however, is not a Pareto-improvement over R , because for arbitrary values of i , R_i can be calculated with less effort than R'_i , because of the lack of recursion (this property is sometimes referred to as “random access”, where “random” has a slightly different meaning, which may be confusing).

The randomness of both R and R' ultimately depends on the randomness of K_0 . However, given K_0 and i , the output is consistent. Such “keyed” random sources are called *pseudorandom number generators*, or PRNG.

Sometimes, guarding any particular small secret (such as K_0) for a long period of time is infeasible. In this case, we resort to *real random numbers*, which depend on the secrecy of other parameters as well. Observe that including additional variables in the input of the one-

way function cannot deteriorate randomness (assuming the one-way function to be a random oracle). Of course, including some inputs may have associated costs.

Typically, the following inputs are included in real random number generators: a counter of queries, to avoid short cycles, the time of counter initialization, to avoid repeated outputs after hardware reset, external inputs of the device outside of the attackers control or prying eyes. Dedicated hardware exists that generates additional input from noisy resistors, semiconductors and radioactive decay. In general, the more independent inputs, the better.

In some personal computer applications and operating systems, the state of the random pool (the input of the one-way function, that is) is sometimes saved on disk in order to preserve randomness between reboots. If this is just an additional input, it can't hurt (again, assuming a random oracle for the hash function). If the saved random seed is the only input (the state of a PRNG, that is), it provides an additional avenue of attack.

4.3 Buying, Selling and Pricing Randomness

If we focus on the costs imposed on attackers, with all things (such as costs to self) being equal, everybody should generate random values for oneself. However, humans are very bad at generating random values (most of us are hopelessly predictable), so we must resort to using machines.

Building machines for ourselves may not be worth the random value, so we usually resort to using machines built by others. And this raises a host of other concerns: can the attacker collude with the provider of the random generator? How do I know that my random numbers are indeed random? These questions are the more important, as real random number generators cannot be black-box tested for maliciously inserted back-doors (predictability by an attacker knowing something that we do not, that is).

The problem is further aggravated, when random numbers must be shared with other parties, as in the case of common RFID tags and pre-paid scratch-off cards — the manufacturer. How to motivate the manufacturer to keep the secret from adversaries? How do we discover information leaks? How much does it cost?

Suppose, each scratch-off card is sold for some p price, the manufacturing costs are $c \ll p$ (typical values would be $c = \$0.15$ vs. $p = \$5$). The total number in one batch is N (a typical value would be $N = 10000$). Thus, the value of the order is Nc , while the value of the secret is Np . It is not at all clear that reputation and opportunity for future business are enough to keep the manufacturer honest.

The solution is using only a fraction of the random values, and not telling the manufacturer, which cards are used and which ones are withheld. Thus, random numbers become random, to some extent, for the manufacturer as well.

Suppose, random numbers are activated as they are sold (the typical model here is a *Poisson-process* with rate λ). There is also a time period (typically modelled by an exponential distribution with an expected value τ) while the activated number remains valid. Thus, there are some valid codes (the expected number is $\tau\lambda$ in this model) that can be “stolen” by the manufacturer. If, at a given time $n \leq N$ codes have been sold, there are $N - n$ codes, that are known to the manufacturer, have not been validated, nor invalidated. If an attempt of using such codes is registered, the manufacturer will get caught, losing future business and the sales from this batch will be stopped. Registering attempts of using already invalidated codes is meaningless, as it can be done by anyone. Hence, by trying to use a stolen code that has not been invalidated, the attacker will succeed with probability $s_n = \frac{\tau\lambda}{N-n+\tau\lambda}$.

For the manufacturer, the expected gain from an attack is $G_n = s_n p - (1 - s_n)C$, where C is the cost of losing future business. G_n increases with n with $G_N = p$. As long as it is negative, the manufacturer has effective incentives not to steal. Let $0 \leq M < N$ such that $G_M < 0$ but $G_{M+1} \geq 0$ (of course, M depends on C). In order to keep the manufacturer honest, from a batch of N codes, we can sell only M , thus $(N - M)c$ must be paid for randomness or $\frac{N-M}{M}c$ for each code sold.

4.4 Exercise

In the above example, we considered only one batch. What happens, if batches of codes are purchased regularly from the manufacturer to keep up with the demand (a Poisson process with rate λ)? Assume an interest rate of I . In this case, it is possible to determine the optimal batch size, to estimate C , etc.

Using reasonable estimates for p , c , τ , λ , and I , give an estimate how much your cellular provider is paying for the randomness of each top-up card.

5 RSA Cryptographic Primitive

“With one skilled at defense, the enemy does not know where to attack.”

— Sun Tzu: Ping Fa (“The Art of War”, Denma Translation)

5.1 Introduction

This cryptographic primitive, first proposed in 1978 [13], was the first one to achieve a complete public key cryptosystem envisioned in [11], useful both for public-key encryption and digital signatures. The underlying hard problem is that of factoring composite numbers with large prime factors. Strictly speaking, it has not been proven to be necessary for breaking RSA, but it is obviously sufficient as we shall see.

The primitive consists of a private and a public transformation that are inverses of one another (see Section 5.3 for more details). For encryption, one performs the public operation, so that only the owner of the private key can calculate the original message by performing the private operation. For a digital signature, one calculates the private operation so that everyone (in possession of the public key) can verify it.

The two operations are modular exponentiations with a modulus n which is a product of at least two large distinct primes (denoted, henceforth, by p_1, p_2, \dots), where the exponents e and d (public and private, respectively) satisfy $ed \equiv 1 \pmod{\phi(n)}$ (note, furthermore, that e and d must be relatively prime with $\phi(n)$). Clearly, $\phi(n) = (p_1 - 1)(p_2 - 1) \dots$. Because of Euler’s theorem (see the mathematical supplement), for any m and c such that

$$m^e \equiv c \pmod{n}$$

we have

$$c^d \equiv m \pmod{n}.$$

This statement is known as *RSA Fundamental Theorem*. Finding d given e and m is known as *RSA Problem*.

5.2 On the Difficulty of the RSA Problem

For moduli with two prime factors, it is not difficult to see that factorization, computing ϕ and solving the RSA Problem are equivalent. In the general case, one can prove that factorization implies ϕ , which, in turn, implies solving the RSA Problem, but it is conjectured that the RSA Problem is no easier than factoring the modulus. Hence, in this section we shall focus on what is known about the difficulty of factoring the modulus.

The computational effort required for most straightforward algorithms (including the brute force approach) used for finding non-trivial divisors depends on the magnitude of the smallest prime factor. Thus, given the magnitude of the product, the greatest cost on attackers utilizing such algorithms is incurred with *two* prime factors in the *same* order of magnitude.

WORK IN PROGRESS ...to be continued...

5.3 Mathematical Supplement

Euler's theorem: Let x and n denote two integers that are relative primes. Then

$$x^{\phi(n)} \equiv 1 \pmod{n}.$$

The theorem follows directly from *Fermat's Little Theorem*: For any integer x and prime p

$$x^p \equiv x \pmod{p}.$$

6 Digital Signatures Based on Discrete Logarithms

“The shih is like drawing the crossbow
the node is like pulling the trigger.”
— Sun Tzu: Ping Fa (“The Art of War”, Denma Translation)

6.1 Overview

Digital signatures based on the difficulty of calculating the discrete logarithm in some finite cyclic group G of order q are constructed from zero-knowledge proofs of discrete logarithms in G . What makes such schemes possible is the (additive) homomorphic and one-way properties of exponentiation (see Section 3.3). The basic idea is to take three values, x – the private key of the signer, M – a message representative (typically the hash value of the actual message) and k – a random value, and subsequently prove, using one-way images, the ability to solve a (modular) linear equation involving x , M and k of which only M is public.

All such algorithms for signing an arbitrary message M involve picking some random integer $k \in \mathbb{Z}_q$, committing to k by calculating the k^{th} element in G (following the multiplicative group notation we write g^k , where g is a generator element in G) and then using M to form a challenge and, finally, calculating the zero-knowledge proof s of knowing the discrete logarithm $x \in \mathbb{Z}_q$ (the *private key*) of some $g^x \in G$ (the *public key*). The verification requires $y = g^x$ – the public key of the signer, M , s and the challenge. The latter two constitute the digital signature of M by the entity with the public key y . The authenticity of y needs to be established separately.

In all such signature schemes, the disclosure of k corresponding to a particular signature leads to the disclosure of the private key used for calculating that signature; the unique solution of the linear equation used for the signature. Similarly, the signature of two different messages using the same value of k also causes the disclosure of the private key. Thus, k must be a collision-free random value (i.e. at least twice the size of a symmetric key). It is important to emphasize that the randomness of k is the Achilles heel of such signature schemes, and therefore a typical target of attacks.

6.2 On the Difficulty of the Discrete Logarithm Problem

It is necessary, albeit not sufficient, for the difficulty of discrete logarithms that the order q of G have at least one large prime factor, to thwart attacks by exhaustive search of the discrete logarithms modulo all prime factors of q and then using the Chinese Remainder Theorem to calculate the discrete logarithm in G . To prevent existential forgery with adaptively chosen plaintext, q should be large enough not only to prevent full enumeration but to provide collision resistance as well. Thus, it should be at least the size of a collision-free hash function (or twice the size of a symmetric key).

The computational cost of computing discrete logarithms in G depends not only on the order of G but also on its representation. Traditionally, the multiplicative groups of $GF(\text{mod } p)$ and $GF(\text{mod } 2^n)$ or subgroups thereof have been used, where p is a large prime and 2^n is a large power of 2. It has to be noted, however, that using $GF(\text{mod } p)$ is usually a pareto-improvement over $GF(\text{mod } 2^n)$, because due to the ease of computations with the latter on binary computers, equivalent strength can be achieved with longer numbers, thus increasing the RAM costs. The only advantage of $GF(\text{mod } 2^n)$ is the reduced ROM cost for firmware implementations on low-end microcontrollers. With today's technology, it is hardly relevant.

With this choice, the difficulty of the discrete logarithm problem requires further considerations in addition to those regarding the largest prime factor of the order of G . Namely, the value of p (or 2^n) should be large enough to thwart the calculation of discrete logarithms using advanced sieve methods (a.k.a. index calculus). The required size is the same as that of the modulus of RSA for equivalent strength against such attacks. Given the size of p , the value of the largest prime factor of q is greatest, if p is a *Sophie-Germain prime*, that is $p' = (p - 1)/2$ is a prime as well. In this case, with a choice of g such that $g \neq p - 1$, the order q of g is divisible by p' . However, similarly to the case of choosing the modulus of RSA to be the product of two equally large primes, the security of such a choice of G is not balanced against attacks depending on the size of p and those depending on the largest prime factor of q , incurring unnecessary costs in calculations in \mathbb{Z}_q .

To balance the security of G against these two kinds of attack, q is typically chosen to be a prime of the size of a collision-free hash function, p is chosen to be a large enough prime such that $q \mid p - 1$ and g is any integer such that $1 < g < p - 1$ and $g^q \equiv 1(\text{mod } p)$. The cryptographic use of such groups has been proposed by Claus-Peter Schnorr [15], hence they are often referred to as *Schnorr-group*.

Another, increasingly popular choice for G is the additive group over the points of non-singular elliptic curves over some finite field (typically $GF(\text{mod } p)$ or $GF(\text{mod } 2^n)$). There are no known attacks that are better than enumeration modulo the prime factors of q , thus the representation of the elements of the group (two coordinates in the finite field of choice) can be much shorter than in the traditional case, thus decreasing the costs of calculations in G . Note,

however, that most research regarding elliptic curves from this perspective is relatively recent, and there are no theoretical reasons to believe that more efficient methods for finding discrete logarithms in such groups will not be developed in the future. Thus, the conservative choice is to resort to elliptic curves only if the computational constraints of the chosen platform and the security requirements of the application cannot be met using other means.

6.3 Possible Signature Equations

The following is a fairly general form of signature and verification equations. Most digital signature schemes based on the difficulty of the discrete logarithm problem (with the notable exception of Schnorr's scheme [15]) are special cases thereof.

The general form of the signature equation is $ak \equiv b + cx \pmod{q}$. The verification is performed in G , using the one-way images: $g^{ka} \stackrel{?}{=} g^b g^{xc}$, where the verifier knows g^k and g^x but not k and x .

Parameters a, b, c denote values from any row of the following table in any order:

$\pm r'$	$\pm s$	m
$\pm r'm$	$\pm s$	1
$\pm r'm$	$\pm ms$	1
$\pm r'm$	$\pm r's$	1
$\pm ms$	$\pm r's$	1

The actual digital signature is the pair (r, s) . One possibility is to set $m = M$, $r = g^k$ and $r' = r \pmod{q}$.

Alternatively, one can set $r = r' = (g^k \pmod{p}) \pmod{q}$. In this case, the signature equations remain as above. The verification equation becomes $r \stackrel{?}{=} ((g^{u_1} y^{u_2}) \pmod{p}) \pmod{q}$, where $u_1 = a^{-1}b \pmod{q}$ and $u_2 = a^{-1}c \pmod{q}$.

If message reconstruction is required, like in the case of RSA signatures where anyone can obtain M from the digital signature itself, then set $r = Mg^k \pmod{p}$ (or $r = (Mg^k \pmod{p}) \pmod{q}$) and set $m = 1$ in the signature equation.

6.4 Case studies

6.4.1 ElGamal

Signature: $r = g^k$, $s = (k^{-1}(M - xr)) \pmod{q}$.

Verification: $r^s y^r \stackrel{?}{=} g^M$.

Historically, ElGamal's signature scheme [14] was the first such construction. The underlying linear equation is $M \equiv xr + ks \pmod{q}$. This corresponds to the first row of the table with $a = s$,

$b = M$ and $c = -r'$.

Observe that the two most expensive calculations in the signature, namely the exponentiation g^k and the modular inverse k^{-1} can be computed independently of M in advance. The message-dependent calculations in the signature are multiplications and additions in \mathbb{Z}_q , which are substantially faster. In particular, much faster than performing the RSA operation. This property is desirable for systems with limited computational resources; most subsequent digital signature schemes based on discrete logarithms inherited this advantage of ElGamal's digital signature.

However, this construction also has some disadvantages, when compared with RSA. With equal public key sizes (which are believed to correspond to equivalent strength), the length of the ElGamal signature exceeds that of RSA. Secondly, it is not possible to recover the message representative from the signature.

6.4.2 Schnorr

Signature: $e = H(M||r)$, $s = (k - xe) \bmod q$, where $r = g^k$.

Verification: $H(M||\hat{r}) \stackrel{?}{=} e$, where $\hat{r} = g^s y^e$.

In the above equations, H denotes a general collision-resistant one-way function with range in \mathbb{Z}_q . The message representative M can be (and indeed usually is) the plaintext of the message itself, not its hash value. Operation $||$ denotes concatenation.

This scheme, proposed by Claus-Peter Schnorr [15] for the purpose of further reducing the computational costs is particularly useful for RFID applications. It is *not* a special case of the general scheme described in Section 6.3, but it is closely related to equation $kM \equiv s + r'x$ also corresponding to the first row of the table with $a = M$, $b = s$ and $c = r'$.

Note that the signature is just twice the size of a collision-resistant hash function, which is a substantial improvement over ElGamal signature and over RSA as well. Furthermore, there is no modular inverse calculation on a per-signature basis, which makes it less expensive than ElGamal's algorithm. Schnorr's algorithm also has the property that no expensive calculations (i.e. modular exponentiation) depend on the actual message to be signed and thus can be performed in advance during idle time.

6.4.3 p-NEW

Signature: $r = Mg^k$, $s = (-k - r'x) \bmod q$.

Verification: $M \stackrel{?}{=} g^s y^{r'} r$.

6.4.4 DSA

Signature: $r = (g^k \bmod p) \bmod q$, $s = (k^{-1}(m + xr)) \bmod q$.

Verification: $r \stackrel{?}{=} ((g^{u_1}y^{u_2}) \bmod p) \bmod q$, where $u_1 = ms^{-1} \bmod q$ and $u_2 = rs \bmod q$.

6.4.5 GOST R34.10-94

Signature: $r = (g^k \bmod p) \bmod q$, $s = (xr + km) \bmod q$.

Verification: $r \stackrel{?}{=} ((g^{u_1}y^{u_2}) \bmod p) \bmod q$, where $u_1 = sv \bmod q$, $u_2 = ((q - r)v) \bmod q$ and $v = m^{q-2} \bmod q$.

7 The Radio Channel

7.1 Introduction

Cryptographic applications in the context of physical security are often using unguided electromagnetic waves for transmitting information. When the wavelength is at least a few centimeters (or, equivalently, the frequency is lower than a few dozen gigahertz), we speak of radio-frequency (RF) communication, which is, by far, the most common and most important case.

In order to be able to design and evaluate such security systems, we will need to know the most important properties of RF communication channels. The aim of this chapter is not to teach RF design, but to highlight the particularities of radio channels that are important for attacks and counter-measures related to cryptography.

The electromagnetic properties of air are very similar to those in vacuum; such electromagnetic fields are *linear*, *isotropic* and *homogenous*: Linear approximations, where the joint field caused by several excitations is expressed by the sum of the fields caused by them individually and the scaling of the excitation by a constant scales the corresponding field as well by the same constant, are accurate over several orders of magnitude. Directions are indistinguishable. Environmental characteristics, such as *permittivity*, *permeability* and *conductivity* are, for all practical purposes, constant all over the space and equal to those in vacuum: $\epsilon = \epsilon_0 \approx 8.85 \cdot 10^{-12} \frac{F}{m}$, $\mu = \mu_0 \approx 1.26 \cdot 10^{-6} \frac{H}{m}$ and $\sigma = 0$, respectively. Thus, Maxwell's equations characterizing the electric and magnetic fields (E and H , respectively) become (with electric and magnetic flux densities $D = \epsilon_0 E$ and $B = \mu_0 H$)

1. $\text{curl } H = \epsilon_0 \frac{\partial E}{\partial t}$
2. $\text{curl } E = -\mu_0 \frac{\partial H}{\partial t}$
3. $\text{div } H = 0$
4. $\text{div } E = 0$

from which it follows that the speed of wave propagation is equal to $c = \frac{1}{\sqrt{\mu_0 \epsilon_0}} \approx 3 \cdot 10^8 \frac{m}{s}$.

7.2 The Hertzian Dipole

Because of the highly linear nature of the electromagnetic field, it is practical to express all excitations in terms of linear combinations of elementary excitations. One such elementary excitation is a short conductive section with a current that is constant over the length of the section, changing as a harmonic function of time t with a frequency f and amplitude I : $i = I \cos(2\pi ft + \phi)$. Obviously, its length l must satisfy $l \ll c/f$. This excitation element is called Hertzian dipole after Heinrich Hertz.

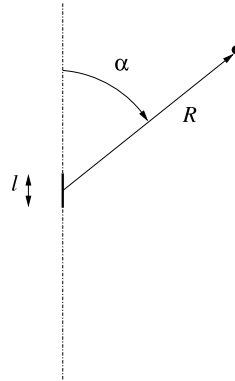


Figure 7.1: Hertzian dipole in a polar system of coordinates

The electromagnetic field excited by the Hertzian dipole has a rotational symmetry around the dipole's axis. In particular, the magnetic field is such that only the tangential component is non-zero (see also Figure 7.1):

$$H_{\tan}(R, \alpha, t) = \frac{l}{4\pi} I \sin \alpha \left(\frac{1}{R^2} \cos(2\pi ft - Rc + \phi) + \frac{2\pi f}{cR} \sin(2\pi ft - Rc + \phi) \right)$$

Observe that the distance-dependence of the field has two components: one vanishing with R^2 called *near-field* and one vanishing with R called *far-field*. Strictly speaking, these two are not electromagnetic fields, as they do not satisfy Maxwell's equations; only their sum does. Yet, from an engineering point of view, the two are often treated separately. At any given distance, the ratio of the amplitude of the far-field to that of the near-field is proportional to the frequency; the higher the frequency, the stronger the far-field radiation. The two amplitudes are equal where $R = \frac{1}{2\pi} \cdot \frac{c}{f}$, which implies that the near-field is dominant only to a fraction of the wavelength. This fact admits of the approximation that the near-field phase is independent of location; the pulsating near-field. To further simplify calculations, we often approximate the far-field locally with plane waves. Since in practice any antenna can be constructed from Hertzian dipoles and any signal can be expressed in terms of harmonic components, the notions of near-field and far-field apply in most cases, albeit often in a less simple manner.

7.3 Restricted Reception

One way to impose additional costs on the eavesdropper is to lower the amplitude of the signal at the eavesdropper's possible locations. Complementing this effort, it is important to take measures that protect locations with high signal amplitude against unauthorized access.

7.3.1 Shielding

One way to confine RF signals is the *Faraday cage* (a.k.a. Faraday shielding in North America or Faraday screening in the UK and Australia), that works at a wide range of wavelengths and geometric configurations. Ideally, this is a sufficiently thick conductive sheet forming a continuous closed surface that separates the protected assets and the attacker. However, this is rarely practical. For one reason or another there are usually some holes in the shielding surface; as the name of the Faraday cage suggest, the whole surface can be a mesh.

By sufficiently thick, we mean that the thickness substantially exceeds the *skin depth* $\delta = \sqrt{\pi f \mu \sigma}^{-1}$ of the conductor. By conductor, we mean that the conductivity σ satisfies $\sigma \gg 2\pi\epsilon f$. Note that both definitions are frequency-dependent. For instance, metals are conductors for the whole practical range of frequencies, while soil cannot be considered conductive for gigahertz-range frequencies and beyond. In the table below, we give the skin depths in copper corresponding to various frequencies:

$f =$	50Hz	1kHz	1MHz	1GHz
$\delta =$	9.4mm	2.1mm	67 μ m	2.1 μ m

Small holes, where the diameter is substantially smaller than the wavelength, are essentially irrelevant, though it must be kept in mind that the same hole may be small for one frequency and large for another. Many small holes should be treated by incorporating their effect into the average conductivity of the shielding material.

Large holes act as secondary sources on their boundary with their respective near- and far-fields, while far-field radiation from the inside passes through the middle of the hole. In general, it is difficult to fully characterize the RF signal leaving a hole because of the various reflections inside the shielding. Formal treatment of holes in the shielding is clearly beyond the goals of this course.

Finally, it is important to emphasize that wave propagation within a hollow conductor (that is within a reflective shielding) is the superposition of all the excitations inside and their reflections. It is easy to see that the lossless reflection model does not apply, as it would result in infinitely increasing signal amplitude within the cage. In reality, the finite conductivity of the shielding results in heat from the induced skin currents. Yet, signal amplitudes within Faraday cages are typically much larger than what the same excitation would cause in the open.

7.3.2 Far-field Transmission

Near-field reception and far-field reception are quite different in how they affect the transmitter. Far-field radiation draws power from the transmitter whether or not it is received; power density decreases with R^2 simply because the total surface of the wavefront increases with R^2 ; the total power carried by the wave does not change. Near-field radiation draws power only if there is a near-field coupling with some other system.

The total power of far-field transmission can be calculated using a quantity called *radiation resistance* (which we denote by R_{rad}) as $P_{\text{rad}} = \frac{1}{2}I^2R_{\text{rad}}$. The efficiency of an antenna is characterized by the ratio of radiation resistance to ohmic resistance. For Hertzian dipole

$$R_{\text{rad}} = \frac{4\pi f^2 l^2}{6\epsilon_0 c^3} \approx 790 \left(\frac{l}{c/f} \right)^2.$$

It holds for other antennae as well that R_{rad} is inversely proportional to the square of the wavelength.

The distribution of power density over the spheric wavefront is never uniform. The distribution of power density defined as a function of direction, while not constant, converges very fast as the distance from the antenna increases. The limit of this distribution is called *directional characteristics* of the antenna. Figure 7.2 below shows the cross-section of the directional characteristics of the Hertzian dipole; a doughnut with no hole in the middle.

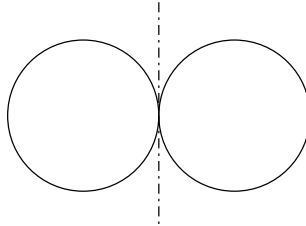


Figure 7.2: Directional characteristics of Hertzian dipoles

Directional characteristics apply only to the far-field. Transmitter and receiver characteristics are the same. The directionality of the antenna is characterized by a quantity called *gain* which is the ratio of the maximal power density over the average power density in the directional characteristics. It is not difficult to see that the gain of a Hertzian dipole is $G = 3/2$.

Another characterization of the directionality is the *effective area*. The relationship between the effective area and gain is the following:

$$A_{\text{eff}} = \frac{c^2}{4\pi f^2} G$$

The ratio of the effective area and the geometric area of the antenna aperture is called *aperture efficiency*. For strongly directed antennas, aperture efficiency is typically between 0.4 and 0.9. From the point of view of security, very high aperture efficiency is not desirable, as such antennas tend to have considerable side lobes (local maxima beside the main beam) in their directional characteristics, which makes them vulnerable to eavesdropping (for transmitters) or jamming (for receivers).

Bibliography

- [1] Gerard Debreu: “Representation of a Preference Ordering by a Numerical Function”, *Decision Process*, pp. 159-165, Wiley, New York, 1954.
<http://cowles.econ.yale.edu/P/cp/p00b/p0097.pdf>
- [2] Bruce Schneier: “Secrets and Lies”, *second edition*, Wiley, New York, 2004
<http://www.schneier.com/book-sandl.html>
- [3] Ian Grigg: “Pareto-Secure” (*unpublished*)
<http://iang.org/papers/pareto-secure.html>
- [4] Nick Szabo: “Trusted Third Parties Are Security Holes” (*essay*)
<http://szabo.best.vwh.net/ttpts.html>
- [5] Mihir Bellare, Phillip Rogaway: “Oracles are Practical: A Paradigm for Designing Efficient Protocols”, *ACM Conference on Computer and Communications Security*, pp. 62–73, 1993
<http://www.cs.ucsd.edu/users/mihir/papers/ro.html>
- [6] Ran Canetti, Oded Goldreich, Shai Halevi: “The Random Oracle Methodology, Revisited” (*preliminary version, unpublished*)
<http://theory.lcs.mit.edu/~oded/rom.html>
- [7] Mitsuru Matsui, Atsuhiro Yamagishi: “A new method for known plaintext attack of FEAL cipher”, *EUROCRYPT 1992*
- [8] Mitsuru Matsui: “Linear cryptanalysis method for DES cipher”, *EUROCRYPT 1993*
http://www.esat.kuleuven.ac.be/~abiryuko/Cryptan/matsui_des.PDF
- [9] Eli Biham, Adi Shamir: “Differential Cryptanalysis of DES-like Cryptosystems” (*lecture notes*), *Weizmann Institute of Science*, 1990
<http://www.cs.technion.ac.il/~biham/Reports/Weizmann/cs90-16.ps.gz>
- [10] Howard M. Heys: “A Tutorial of <http://citeseer.ist.psu.edu/diffie76new.html> Linear and Differential Cryptanalysis” (*on-line tutorial*)
http://www.engr.mun.ca/~howard/Research/Papers/ldc_tutorial.htm

- [11] Withfield Diffie, Martin Hellman: “New Directions in Cryptography”, *IEEE Transactions on Information Theory*, Vol. 22, pp. 644–654, 1976
<http://citeseer.ist.psu.edu/diffie76new.html>
- [12] Withfield Diffie, Martin Hellman, Ralph Merkle: “Cryptographic apparatus and method”, *US Patent #4200770*, 1980
<http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=4,200,770>
- [13] Ron Rivest, Adi Shamir, Len Adleman: “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”, *Communications of the ACM*, Vol. 21, pp. 120–126, 1978
<http://theory.lcs.mit.edu/~rivest/rsapaper.pdf>
- [14] Taher ElGamal: “A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms”, *Proc. CRYPTO 84 on Advances in Cryptology*, 1985
<http://crypto.csail.mit.edu/classes/6.857/papers/elgamal.pdf>
- [15] Claus P. Schnorr: “Efficient Identification and Signatures for Smart Cards”, *EUROCRYPT* 1989
<http://www.springerlink.com/link.asp?id=p151aecakg98mtyd>
- [16] Adrian McCullagh, William Caelli: “Non-Repudiation in the Digital Environment”, *First Monday*, Vol. 5 (8), 2000
http://firstmonday.org/issues/issue5_8/mccullagh/index.html