Cryptography and Beyond

Cryptography and Beyond

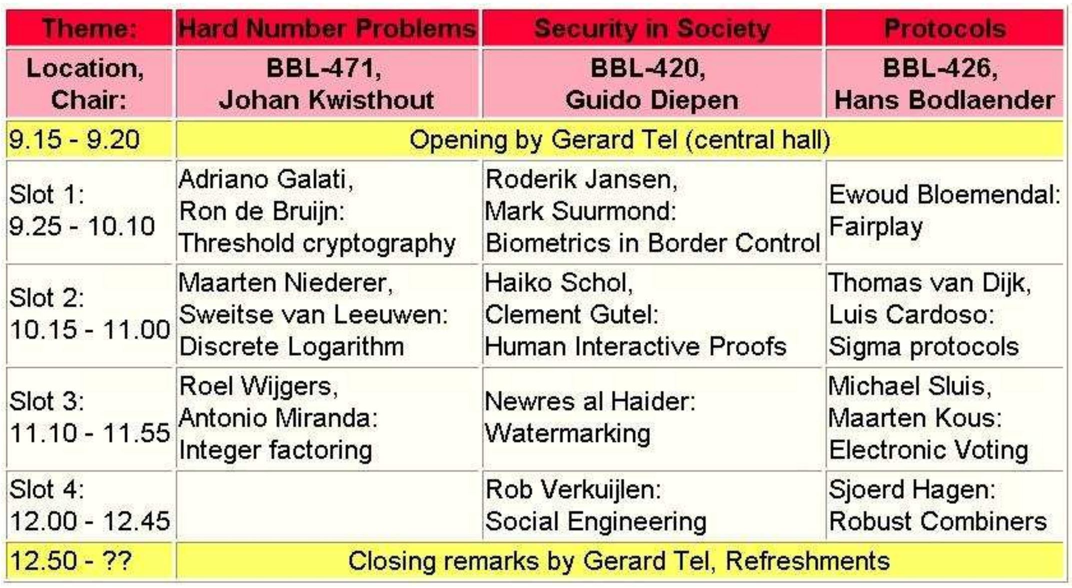
Gerard Tel

Contents

Preface

This syllabus contains papers discussed at a small sympusium called *Cryptography and Beyond*, held on January 31, 2006, at Utrecht university. Presentation at the symposium and contribution to this syllabus were obligatory requirements in the course *Cryptography* of the Master of Algorithmic Systems.

Eleven presentations were given on the Symposium, arranged in three tracks as indicated in the schedule:



Theme:	Hard Number Problems	Security in Society	Protocols
Location, Chair:	BBL-471, Johan Kwisthout	BBL-420, Guido Diepen	BBL-426, Hans Bodlaender
9.15 - 9.20	Opening by Gerard Tel (central hall)		
Slot 1: 9.25 - 10.10	Adriano Galati, Ron de Bruijn: Threshold cryptography	Roderik Jansen, Mark Suurmond: Biometrics in Border Control	Ewoud Bloemendal: Fairplay
Slot 2: 10.15 - 11.00	Maarten Niederer, Sweitse van Leeuwen: Discrete Logarithm	Haiko Schol, Clement Gutel: Human Interactive Proofs	Thomas van Dijk, Luis Cardoso: Sigma protocols
Slot 3: 11.10 - 11.55	Roel Wijgers, Antonio Miranda: Integer factoring	Newres al Haider: Watermarking	Michael Sluis, Maarten Kous: Electronic Voting
Slot 4: 12.00 - 12.45		Rob Verkuijlen: Social Engineering	Sjoerd Hagen: Robust Combiners
12.50 - ??	Closing remarks by Gerard Tel, Refreshments		

I hope that the reader will get an impression of what we did in the course and in the symposium.

Gerard Tel, March 2006.
email: `gerard@cs.uu.nl`

Chapter 1

Discrete Logaritm

Written by *Maarten Niederer and Sweitse van Leeuwen.*

This chapter covers the discrete logarithm and related problems. We will introduce the discrete logarithm and treat three important algorithms to find discrete logarithms.

The second part describes the use of Elliptic Curves and Braid groups in cryptography. We will introduce these groups and explain how they could be used for encryption. At the end of this chapter we will discuss their advantages and disadvantages over the multiplicative group.

1.1 Introduction to the discrete logarithm

Several cryptographical algorithms use some calculation in a group. We will focus on the multiplicative group $\mathbb{Z}_p^*$ for some fixed prime p. An abelian group is defined by the triple $(V, e, \circ)$, with V a set, e the unit element and $\circ : V \times V \to V$ an infix operator. Any elements $a, b, c \in V$ satisfy:

$$
\begin{aligned}
a \circ b &\in V && (1.1)\\
a \circ e = e \circ a &= a && (1.2)\\
\exists a' : a \circ a' &= e && (1.3)\\
a \circ (b \circ c) &= (a \circ b) \circ c && (1.4)\\
a \circ b &= b \circ a && (1.5)
\end{aligned}
$$

For an easy introduction to groups, read Armstrong [Arm88]. Our choice $V = \mathbb{Z}_p^*$ does satisfy those restrictions. We will denote $h_g(x) := x \circ g$, for $g, x \in V$, multiplication by g. Now we can speak of exponentiation of an arbitrary number in V.

$$g^k := h_g^k(e) \quad \text{for} \quad g \in V, k \in \mathbb{N}$$

```
function $exponentiate(a, k)$:
if $k = 0$ return $e$;
    if $k < 0$ return $exponentiate(a, -k)$;
    if $k$ is odd return $exponentiate(a, k) \circ a$;
    $b := exponentiate(a, \frac{1}{2}k)$;
    return $b \circ b$;
```

Algorithm 1.1: Fast exponentiation algorithm

The naive implementation involves k iterations of the group function h_g. However the exponentiation can be limited to $O(\log k)$ group iterations, by algorithm 1.1.

Since $\log k$ is the number of bits representing k, if the group operation can be calculated in polynomial time, the exponentiation can also be calculated in time polynomial in the number of bits of k. We will assume that group operations can be calculated in polynomial time.

In the real world, at least in $\mathbb{R}$, there is an inverse operation of exponentiation, which we call the logarithm. In $\mathbb{R}$ the logarithm is defined as:

$$\log_g z = k \quad \text{if} \quad g^k = z$$

Since we have an exponentiation in our group, we can extend the definition of the logarithm in $\mathbb{R}$ to any group. We call the logarithm in a group the discrete logarithm.

From the definition of the logarithm, there doesn't follow any intuitive fast algorithm. In $\mathbb{R}$ the logarithm is calculated numerically, in this way the logarithm can be found pretty accurate with a small amount of computation power. Since we are never able to compute in $\mathbb{R}$ with infinite accuracy this is probably the best we can do, and the approximation is mostly good enough for its purpose.

Taking the discrete logarithm in a group is an entirely different case. When the discrete logarithm exists, we know that there is an exact solution in $\mathbb{N}$ for the exponent k. Hence k can be represented exact. Also approximation will not work since $g^{k \pm \delta}$ can't be calculated for $\delta \notin \mathbb{N}$.

In 1997, Shoup proved that any algorithm that is based only on group operations has an algorithmic complexity of at least $O(\sqrt{|V|})$. Hence an efficient algorithm only using group operations is not available.

1.1.1 ElGamal encryption

ElGamal cryptography is an example of *public-key cryptography*. The ElGamal algorithm is based on a cyclic group G of big prime order and with generator g. The private key is an element a of this group, the public key is given by $b(a) = g^a$.

The security of ElGamal depends on the feasibility of calculating the discrete logarithm. Anyone who is able to compute discrete logarithms can deduce the private ElGamal key from the public one simply by computing $\log_g b$. Thus it is important to know that the discrete logarithm can not be calculated.

1.2 Algorithms for finding the discrete logarithm

There are several ways to find the discrete logarithm. In this section we will discuss three of them. The first two are group algorithms, which means that the algorithm works for any underlying group. The third algortihm uses the special properties of $\mathbb{Z}_p^*$.

1.2.1 Shanks' method

Theory Let g be a generator of a group G with big prime order p. Then for every $b \in G$ there is an a such that $g^a = b$. Shanks' method searches for an intersection between two sets of size $\sqrt{p}$ and calculates the value of a from this.

Start with choosing two numbers m and t such that $m \cdot t > p$ and construct the set

$$M = \{g^{jt} \mid 0 \leq j < m\}$$

Now we know that for every element b of G, there is a number $0 \leq i < t$ with $b \cdot g^i \in M$. In particular this holds for $b(a) = g^a$, so if we construct the sequence $b_i := \{b \cdot g^i\}_{i=0}^{t-1}$, one of the elements is contained in M. If we have found i_0 and j_0 with $b \cdot g^{i_0} = g^{j_0 t}$, then $b = g^{j_0 t - i_0}$ and hence $\log_g b = a = j_0 t - i_0$.

Instead of using the full group G, you can also use a cyclic subgroup $H = <h>$ of G. The complexity of discrete logarithm in H is the same as in G. If we want to find $\log_h b$ for $b \in G$, then the intersection of the sequence b_i and M is empty if and only if $b \notin H$.

Implementation The algorithm for Shanks' method follows the theory quite exact, see Algorithm 1.2. The choice of the parameters m and t is important for the memory usage and the time needed for the calculation. Building the set M takes $O(m)$ time and memory. In the next step we will do a lot of *find* operations of this set, so it is useful to arrange the set in such a way that we can quickly perform this operation. Therefore we use a hashtable and store the matching value of j with each entry g^{jt}.

In the second part of the program we calculate the sequence b_i and look for each term in the hashtable. In worst case we have to test all values up to t, whereas the expected number of test is $\frac{t}{2}$, both of which take $O(t)$ time. If b_i is contained in M, get the corresponding value of j from the table and then proceed as described above.

The time needed for the execution of this algorithm is $O(m + t)$. When we choose m and t about as large as $\sqrt{p}$, this expression reduces to $O(\sqrt{p})$.

1.2.2 Pollard's Rho method

We need a huge amount of memory when we want an optimal performance of Shanks' algorithm. Even for our small tests the necessary amount of memory was just available in main-memory. For real calculations, even secondary storage on clusters of computers would not suffice. We will now discuss a method that uses a constant amount of memory [Pol78].

```
function $shank(g, p, b)$
    $m := \sqrt{p}; t := \sqrt{p};$
    $M :=$ new $Hashtable;$
    for $j := 0$ to $m$
        $M.put(g^{jt}, j);$
    for $i := 0$ to $t$
        $j_0 := M.get(b \cdot g^i)$
        if $j_0 \neq null$
            return The logarithm of $b$ is $j_0 \cdot t - i$
    return $b$ is not a power of $g$
```

Algorithm 1.2: Shanks' algorithm

Theory In Pollard's Rho method we need a fixed group function that behaves somewhat random. Let H be a hashfunction $V \rightarrow \{1, 2, 3\}$, then the following function suffices:

$$f(z) = \begin{cases} z \circ y & \text{when } H(z) = 1 \\ z \circ g & \text{when } H(z) = 2 \\ z \circ z & \text{when } H(z) = 3 \end{cases} \tag{1.6}$$

Here y is the number of which we want to take the discrete log and g is the base of the discrete log. Since f randomizes enough we can assume for random $x, y \in V$:

$$\forall_{x \neq y} : P(f(x) = f(y)) = \frac{1}{|V|}. \tag{1.7}$$

Now Pollard's rho method is about developing 2 sequences by iterating f. When the elements of both sequences are equal we speak of a collision and we can calculate the discrete logarithm we are looking for, we will show this later.

At first both sequences t and z need to have starting points, we will take $z_0 = t_0 = y^n \circ g^m$ for random n, m. Now we develop the z sequence twice as fast, one iteration will become

$$\begin{aligned} z_{i+1} &= f(z_i) \\ t_{i+1} &= f(f(t_i)) \end{aligned}$$

In effect $t_i = z_{2i}$. Since both sequences are limited to V and V is limited of size there will be a smallest constant k such that $\exists_j, \forall_{i \geq j} : z_i = z_{i+k}$. When $i = kl \geq j$ with $l \in \mathbb{N}$ then $t_i = z_{2i} = z_{i+kl} = z_i$ and we have the collision of both sequences. By 1.7 we are expected to have a collision after $O(\sqrt{|V|})$ step.

On collision we have $z_k = y^a \circ g^b$ and $t_k = y^c \circ g^d$ for some values of a, b, c, d. Since $z_k = t_k$, we find

$$y^a \circ g^b = y^c \circ g^d \tag{1.8}$$

```
function $pollardRho(g, y)$
    repeat
        $a := random;$
        $b := random;$
        $z := y^a \circ g^b;$
        $(t, c, d) := (z, a, b);$
        repeat
            $(z, a, b) := F(z, a, b);$
            $(t, c, d) := F(F(t, c, d));$
        until $z = t$
    until $(a - c)$ is invertible
    return $(d - b) \circ (a - c)^{-1}$
```

Algorithm 1.3: Pollard's Rho algorithm

Subsituting $y = g^x$ in equation 1.8 enables us to find the discrete logarithm:

$$\begin{aligned} g^{ax} \circ g^b &= g^{cx} \circ g^d \\ g^{ax+b} &= g^{cx+d} \\ x &= \frac{d-b}{a-c}. \end{aligned}$$

In most cases $a - c$ has an inverse and we can compute x if we had only kept track of a, b, c and d. If $a - c$ has no inverse, we will restart our algorithm. Keeping track of a, b, c, d number is not a problem. We know n, m which are the starting values of a, b and c, d. Also at each iteration of f we know what happens to the values of a, b, c, d. This results in algorithm 1.3.

In this algorithm function F is computed as:

$$F(z, a, b) = \begin{cases} (z \circ y, a+1, b) & \text{when } H(z) = 1 \\ (z \circ g, a, b+1) & \text{when } H(z) = 2 \\ (z \circ z, 2a, 2b) & \text{when } H(z) = 3 \end{cases} \tag{1.9}$$

1.2.3 Index calculus method

Both precedent methods use only that the calculation are to be carried out in a group. When we would use only the group properties, Shanks' and Pollard's method are among the fastest algorithms.

However in practice the group can have more properties than just the group properties. For example, in practice the group $\mathbb{Z}_p^*$ for p prime is used. The elements in the group can also represented as element of $\mathbb{Z}$. In this representation, the elements have more properties we can exploit.

Theory At first we assume that g is a true generator of $V = \mathbb{Z}_p^*$. The index calculus algorithm consists of three steps, in the first two steps a table is constructed which is consulted in the last step. The first two steps are fixed for fixed p, g and variable y. While the last step is variable for variable y. The last step is a relatively fast part of the algorithm.

The first step consists of searching for relations. A relation is a number in V that factorizes over small prime numbers only. We take only the first b prime number for this purpose, then p_b is the value of the b^{th} prime. The chance that a random number in V is a relation can be estimated by

$$\frac{1}{p} p^{1-\frac{\log\log p}{log p_b}}$$

Increasing b gives you a higher chance of a relation, while increasing p lowers this chance.

In the first step we search for at least b relations of the form g^k for random k. At the end of the first step we will have (at least) b equalities like

$$g^k = \prod_{j \in \{1,\ldots,b\}} {p_j}^{v_j}$$

Since g is a generator, we can express each p_j as g^{t_j}, for some t_j and the equations can be expressed in a linear form

$$k = \sum_{j \in \{1,\ldots,b\}} t_j \circ v_j$$

The first step results in a system of linear equations. The second step simply consists of solving this system. At the end we will have calculated the indexes of the first b primes, the sequence t_j.

After completing step 2 we will have the indexes of a large number of prime numbers. The last step only consists of finding a random k for which $g^k \circ y$ is a relation. When this k is found, we have

$$g^k \circ y = \prod_{j \in \{1,\ldots,b\}} {p_j}^{v_j}$$

and

$$k + \log_g y = \sum_{j \in \{1,\ldots,b\}} t_j \circ v_j$$

In this $\log_g y$ is the only unknown variable, which we than can compute.

1.2.4 Comparison of the presented algorithms

In practice we restricted our implementation to the group $\mathbb{Z}_p^*$ for some prime p. We picked a p at random of a certain size, we also picked a g at random. We ensured that $ord(g)$ is in size exactely 4 bits smaller than p. For Shanks' algorithm we chose the size of the table to be $\sqrt{ord(g)}$. For larger cases memory-swapping occured giving a performance degradation.

Size of modulus	**Shanks'**		**Pollard's Rho**	
	memory usage	run-time	succes rate	run-time
36 bits		11 s	100 %	8 s
37 bits		16 s	100 %	9 s
38 bits		20 s	93 %	25 s
39 bits		30 s	100 %	27 s
40 bits		43 s	100 %	31 s
41 bits		69 s	100 %	41 s
42 bits		89 s	87 %	84 s
43 bits		96 s	80 %	173 s
44 bits	182 MB	165 s	100 %	107 s
45 bits		255 s	100 %	109 s
46 bits	365 MB	342 s	93 %	214 s

Table 1.4: Running times of Shanks' and Pollard's algorithm

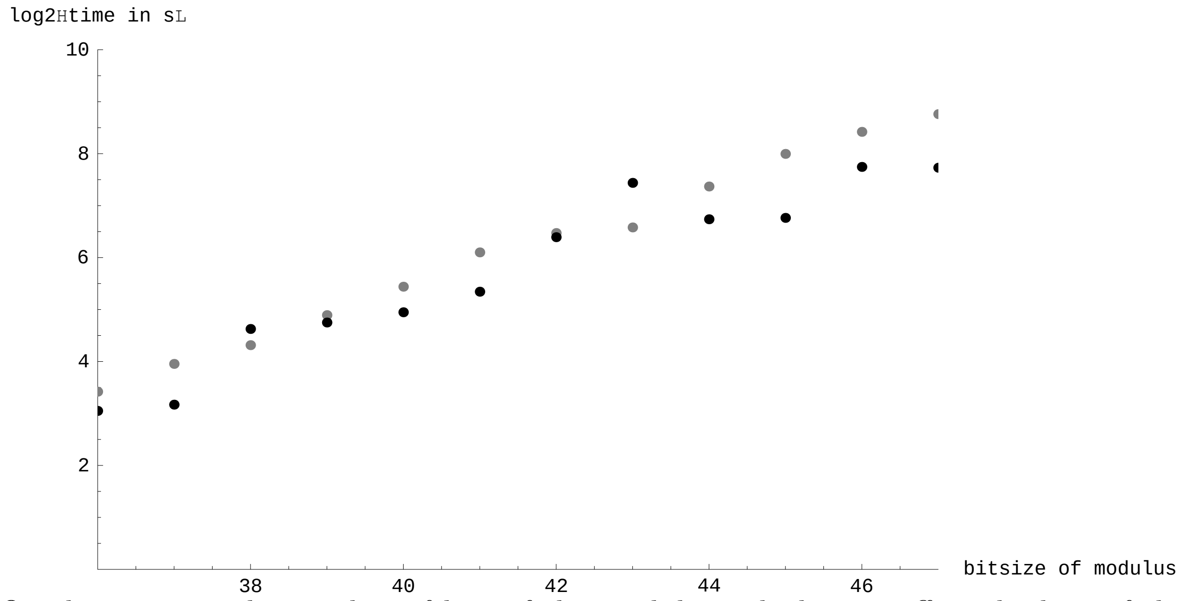
In table 1.4 the measured performances are shown, then results are averages over 15 runs. For the Shanks' algorithm all runs terminated, however Pollard's Rho method did not. We cut off the search whenever 5 consequetive tries failed to result in an invertable $(a - c)$, see Pollard's Rho algorithm 1.3. One can see that several such failures (such as in the 43-bits case), will cause a steep rise in run-time.

In the graph 1.5 we plotted the run-times of both algorithms. Since this is a log-log plot, one can see from the slope of the graph how the algorithm scales. The slope is about $\frac{1}{2}$ which gives good evidence that the algorithms scales $O(\sqrt{q})$ for at least the range $q \in [2^{36}, 2^{46}]$.

We did not implement the index calculus method ourselves, however we found the results of Studholme[Stu02]. Be aware that the calculations are performed in a different computating environment, nevertheless just a single home computer. Also many optimalizations of the index calculus method are applied. The results are shown in graph 1.6. The index calculus algorithm runs much faster then the methods we implemented.

1.3 The discrete logarithm in different groups

As mentioned before, Shoup [Sho97] has proven a lower bound of algorithmic complexity for calculating the discrete logarithm in an abstract group. From a cryptographic point of view, we are very happy with this lower bound. However in the group $\mathbb{Z}_p^*$ we discussed so far the discrete logarithm can be found more efficient. In this section of the paper we will focus on other groups for which hopefully no faster methods than the group based algortihms like Shanks' and Pollard's exist.



On the x-axis is the number of bits of the modulus, which is in effect the $\log_2$ of the modulus. On the y-axis we plotted the $\log_2$ of the average run-time over 15 runs. The black dots are the measurements of Pollard's Rho algorithm, the gray dots of Shanks'algorithm.

Figure 1.5: Shanks' and Pollard's Rho run-times

1.3.1 Elliptic curves

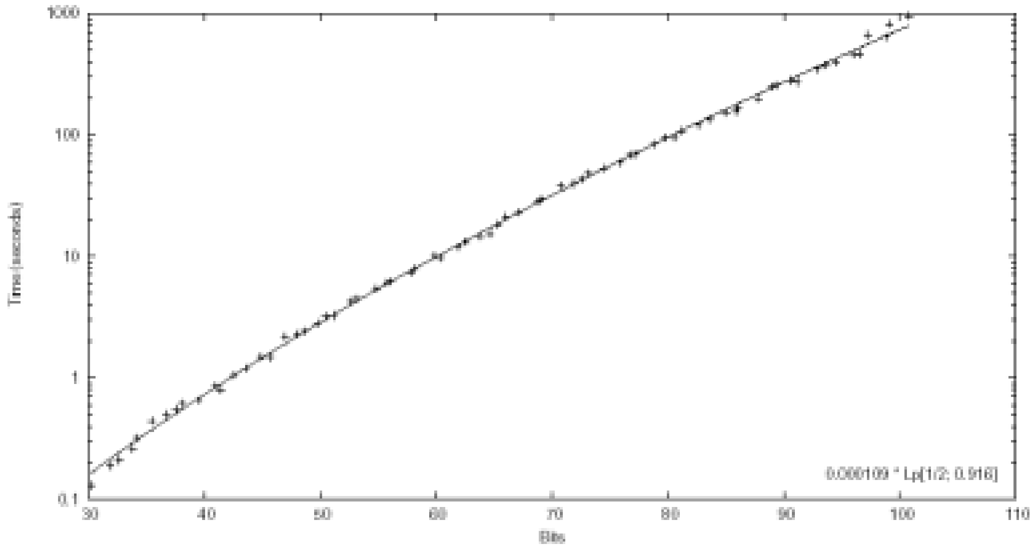
The elleptic curve is a collection of groups parameterized by a, b and q. There are two classes of these groups, one for large primes q, two other with $q = 2^m$, for large m. We will concentrate us on the former case. The set the group acts on is

$$E_{q,a,b} = \{\mathcal{O}\} \cup \{(x, y) \in \mathbb{Z}_q \times \mathbb{Z}_q \mid y^2 \equiv_q x^3 + ax + b\} \tag{1.10}$$

This set is visualized in 1.7. We denote P and Q for arbitrary elements of $E_{q,a,b}$. Whenever $P = (x, y)$, we define its inverse $-P = (x, -y)$, which is also in $E_{q,a,b}$. We denote the group operation by an infix + sign. The size of $E_{q,a,b}$ is bouded by the theorem of Hasse:

$$q - 2\sqrt{q} + 1 \leq \|E_{q,a,b}\| \leq q + 2\sqrt{q} + 1. \tag{1.11}$$

The group is given by the triple $(V, \mathcal{O}, +)$, with $V = E_{q,a,b}$ the set as defined in 1.10, $\mathcal{O}$ the identity and the operation + on $P = (x_p, y_p)$ and $Q = (x_q, y_q)$ defined as follows:



Running time of the index calculus method as a function of the problem size by Studholme[Stu02]. The x-axis shows the problem size in bits, while the y-axis shows the run-time in seconds.

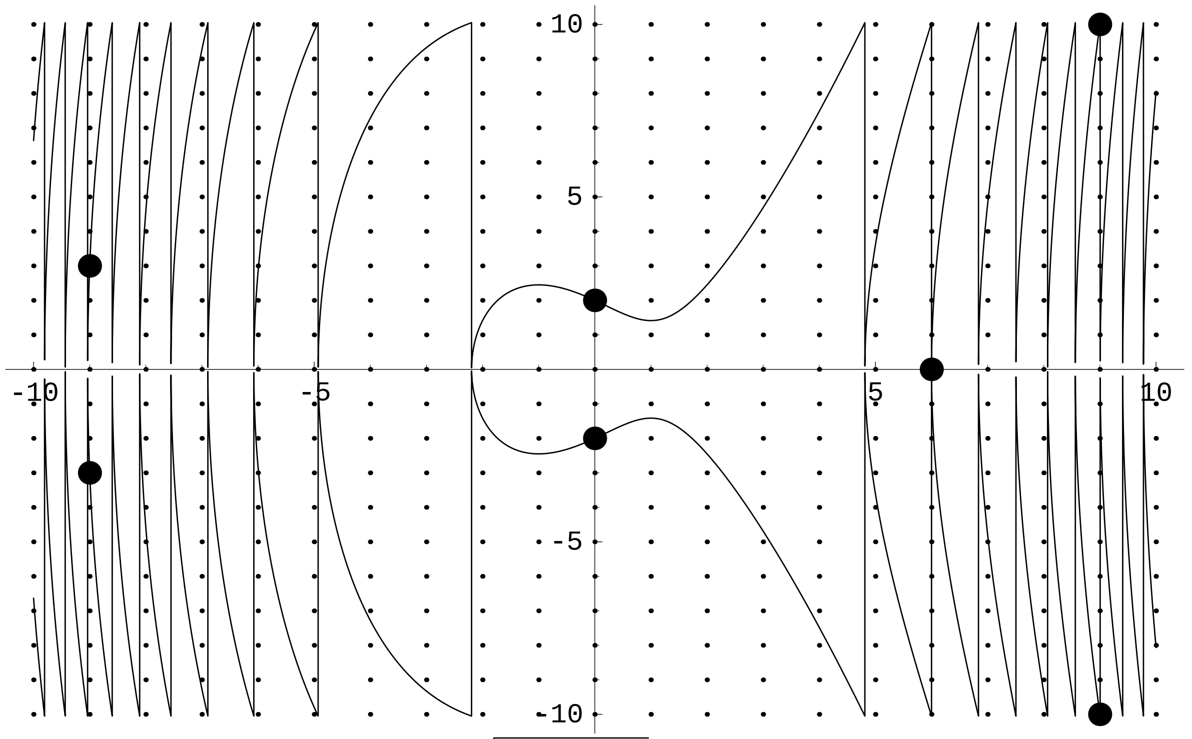
Figure 1.6: INDEX CALCULUS RUN-TIME

$$
\begin{aligned}
\mathcal{O} + P &= P + \mathcal{O} = P \\
-P + P &= \mathcal{O} \\
P + Q &= \mathcal{O} \quad \text{if} \quad P = Q \quad \text{and} \quad y_p = 0 \\
P + Q &= (x_r, y_r) \\
\text{where} \quad & x_r = c^2 - x_p - x_q \\
& y_r = c(x_p - x_r) - y_p \\
& c = \begin{cases} \dfrac{y_q - y_p}{x_q - x_p} & \text{if} \quad P \neq Q \\ \dfrac{3x_p^2 + a}{2y_p} & \text{if} \quad P = Q \quad \text{and} \quad y_p \neq 0 \end{cases}
\end{aligned}
$$

From this we can prove that the proposed group is an abelian group.

The operations also has a very clear geometric meaning. It assings to two points on the curve a third point that is on the intersection of the curve and the straight line through the first two points. However, when both point share the x-coordinate, the straight line won't intersect the curve again. For this reason a point $\mathcal{O}$ is added to the set. $\mathcal{O}$ is defined as the the result of the operation in these cases. Also when $P + P$ is calculated, there is no straight line through the points. In this case the tangent of the curve trough P is used.

Up to this date there are no faster methods known for finding the discrete logarithm in this group then the abstract group algorithms (Pollard's rho and Shanks'). However



The lines are the equations $y = \pm(\sqrt{x^3 + ax + b} \mod q)$ for $a = -3$, $b = 4$ and $q = 101$. The dots are all the points in $\mathbb{Z} \times \mathbb{Z}$ and the larger dots also satisfy $y^2 \equiv_{\text{mod } q} x^3 + ax + b$. In total 95 points in $\mathbb{Z}_q \times \mathbb{Z}_q$ satisfy this condition.

Figure 1.7: ELLEPTIC CURVE VISUALTIZATION

in special circumstances this can be done quite efficient. When $\|E_{q,a,b}\| = q$, an isomorphism $(E_{q,a,b}, +) \rightarrow (\mathbb{Z}_q, +)$ can be found. An isomorphism is a bijection between the sets of both groups that maintains the operation structure. In the group $(\mathbb{Z}_q, +)$ the discrete logarithm is trivial, namely division by the index of the base element g. Another possible attack is the Pohlig-Hellman attack. This attack is possible when g generates a (sub)group of $E_{q,a,b}$, which has a size which has no large prime factors. In effect $\|E_{q,a,b}\|$ needs to have a large prime factor.

Finding parameters q, a, b such that that the system is safe, is not an easy task. At first $\|E_{q,a,b}\|$ needs to be computed (and probably a many number of times). This can be done with the Schoof-Elkies-Atkin Algorithm, see [IAG99], that has run-time complexity of $O(\log^6 q)$. Compare this to taking a random prime, needed for $\mathbb{Z}_p^*$, which runs in $O(\log^4 p)$. Then this value $\|E_{q,a,b}\|$ needs to be factorized, also a difficult task. When we find a large prime factor of $\|E_{q,a,b}\|$, we can find an appropriate generator g pretty efficient.

The search for good parameters takes several hours on a modern PC. But when we are done constructing a curve, we can suffice with much smaller keys and computations than we had to use in the multiplicative group $\mathbb{Z}_p^*$.

1.3.2 Braid groups

The braid groups B_n form a family of groups, which are non-abelian for $n > 2$ and infinite for $n > 1$. They have a clear geometric interpretation and a short algebraic definition. We will start out from the formal definition and give the interpretation together with some examples afterwards.

Definition 1.1 *The* braid group B_n *is the group generated by the elements* σ_1 *up to* σ_{n-1} *with the relations* $\sigma_i\sigma_j = \sigma_j\sigma_i$ *if* $|i-j| > 1$ *and* $\sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j$ *if* $|i-j| = 1$.

We will denote the unit element by e, the group operation is juxtaposition, i.o.w. just place the elements next to each other. Any element of this group can be represented through a string of generators, for example $\sigma_1\sigma_2\sigma_1\sigma_3^{-1}\sigma_4\sigma_1$. This string is called a *word*, the elements are the *letters*. The representation for an element is not unique: the element σ_3 for example can be represented in all of the following ways by making use of the group axioms 1.2 and 1.3 or the relations defined in definition 1.1.

$$\begin{aligned}\sigma_3 &= \sigma_3\sigma_1\sigma_1^{-1}\\ &= \sigma_1\sigma_3\sigma_1^{-1}\\ &= \sigma_1\sigma_2^{-1}\sigma_2\sigma_3\sigma_2^5\sigma_2^{-5}\sigma_1^{-1}\\ &= \sigma_1\sigma_2^{-1}\sigma_3\sigma_2\sigma_3\sigma_2^4\sigma_2^{-5}\sigma_1^{-1}\end{aligned}$$

Fortunately there exists a unique reduced form of each word, called *greedy normal form* [ECH+92]. Reducing a word w to normal form takes linear time for each redundant letter. The worst case time is quadratic in the length of the word. The advantage of this representation is that comparing two words for equality takes just linear time in the length of the word w. The drawback is that applying the group operation to two normal words does not necessary yield a normal word, so you need to reduce it afterwards. The length of this reduced form is called the *complexity* of the word.

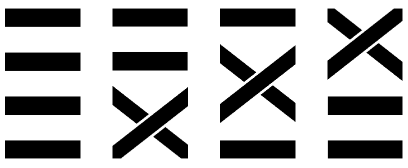
Conjugation problems In $\mathbb{Z}_n{}^*$ the difficulty of finding the discrete logarithm enables us to use the ElGamal encryption scheme, just like the difficulty of integer factorization makes RSA encryption possible. The difficult problem for the braid groups is the *conjugacy problem*. In the following definitions the main concepts of the problem are described.

Definition 1.2 *A pair* $x, y \in G$ *is* conjugate in G *if there exists an* $a \in G$ *such that* $x = a^{-1}ya$.

The *conjugacy problem* is equivalent to the question: "Are x and y conjugate?" If x and y are conjugate, we can ask for an a such that $x = a^{-1}ya$. This is called the *conjugator search problem*. At the end of this section we will discuss the difficulty of this problem.

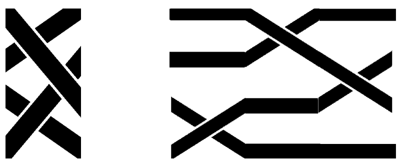
Casus 1.8: GEOMETRIC INTERPRETATION OF BRAIDS

The braid group B_n is the set of all distinct knots of n left to right threads. They can be visualized by two columns of n dots connected by straight lines. It is however not a planar graph but a representation of a 3D-structure, so two threads can cross over or under eachother. The set of generators of B_4 can be represented by the following 4 graphs:



e σ_1 σ_2 σ_3

The generators for B_n are created in the same way. By combining these generators you could create all the possible knots, for example:



$r = \sigma_1 \; \sigma_3^{-1} \; \sigma_2$

1.3.3 Key exchange schemes in braid groups

Braids can be used to exchange keys by public key cryptography. We will discuss the commutative scheme developed by Ko e.a. [KLC$^+$00] and a simplified version of the protocol described by Anshel e.a.[IAG99]. Before we proceed to the scheme of Ko e.a. we will define two mutually commutative subgroups UB_n and LB_n.

Definition 1.3 *The* upper *and* lower braid groups UB_n *and* LB_n *are defined as subgroups of* B_n*:*

$$\begin{aligned} UB_n &= <\sigma_1, \ldots \sigma_{m-1}> \\ LB_n &= <\sigma_{m+1}, \ldots \sigma_{n-1}> \end{aligned}$$

with $m = \lfloor \frac{n}{2} \rfloor$.

If we take $p \in UB_n$ and $q \in LB_n$, then we know by the reduction rules on braid groups (see definition 1.1) that these elements commute: $pq = qp$. Now we are properly equipped for the first key exchange scheme.

Commutative key exchange We assume Alice has a private key $a_A \in UB_n$ and Bob has a private key $a_B \in LB_n$. They share a public key $p \in B_n$. They proceed as described in protocol 1.9. After the exchange they have both the same key

$$k = a_A y_B a_A^{-1} = a_A a_B p a_B^{-1} a_A^{-1} = a_B a_A p a_A^{-1} a_B^{-1} = a_B y_A a_B^{-1} = k$$

Alice	Bob
$y_A := a_A p a_A^{-1}$; send y_A	receive y_A
receive y_B	$y_B := a_B p a_B^{-1}$; send y_B
compute $k = a_A y_B a_A^{-1}$	compute $k = a_B y_A a_B^{-1}$

Protocol 1.9: KEY EXCHANGE BY KO E.A.

because the private keys of Alice and Bob commute. A possible third party could intercept the values of p, y_A and y_B. If he wants to find the key k, he has to solve the following problem: Given p, upu^{-1} and lpl^{-1} with $u \in UB_n$ and $l \in LB_n$, find $ulpl^{-1}u^{-1}$.

Claim 1.4 *This variation on the conjugacy search problem is a difficult problem.*

If we assume that the conjugacy problem is hard on B_n, then solving the problem described above on B_{2n} is as least as hard, because the group B_n is equivalent to the group UB_{2n}.

Anshel-Anshel-Goldfeld scheme In this key exchange the public key is double set of m braids $p_1, \ldots p_m, q_1, \ldots q_m$. The private key is a function mapping m different letters to B_n, for example $f_{ex}(\tau_1, \ldots, \tau_m) = \tau_2 \tau_{m-1} \tau_3^{-1}$. Note that $f(a_A q_1 a_A^{-1}, \ldots, a_A q_m a_A^{-1}) = a_A f(q_1, \ldots, q_m) a_A^{-1}$. If Alice and Bob follow protocol 1.10, they end up with the same key

$$\begin{aligned}
a_A f_A(y_B)^{-1} &= a_A f_A(a_B p_1 a_B^{-1}, \ldots, a_B p_m a_B^{-1})^{-1} \\
&= a_A a_B f_A(p_1, \ldots, p_m)^{-1} a_B^{-1} \\
&= a_A a_B a_A^{-1} a_B^{-1} \\
&= a_A f_B(q_1, \ldots, q_m) a_A^{-1} a_B^{-1} \\
&= f_B(a_A q_1 a_A^{-1}, \ldots, a_A q_m a_A^{-1}) a_B^{-1} \\
&= f_B(y_A) a_B^{-1}
\end{aligned}$$

An attack on this protocol may be easier than the conjugator search problem, because an attacker possesses m pairs of conjugates instead of one. This problem is called the *Multiple Conjugator Search Problem.*

Similar schemes exist for encryption/decryption, authetication and signing. For a brief overview of these schemes, see for example [Deh04].

Security of braid groups The security of braid groups relies on the difficulty of the conjugator search problem and related problem as described above. It is not clear whether these problems are difficult enough. Vladimir Shpilrain points out in his article [Shp04] that we should get rid of braid groups.

Alice	Bob
compute $a_A = f_A(p_1, \ldots, p_m)$	compute $a_B = f_B(q_1, \ldots, q_m)$
compute $y_A = (a_A q_1 a_A^{-1}, \ldots, a_A q_m a_A^{-1})$	compute $y_B = (a_B p_1 a_B^{-1}, \ldots, a_B p_m a_B^{-1})$
send y_A	receive y_A
receive y_B	send y_B
compute $k = a_A f_A(y_B)^{-1}$	compute $k = f_B(y_A) a_B^{-1}$

Protocol 1.10: Key exchange by Anshel, Anshel and Goldfeld

Summary and Conclusions

The multiplicative group allows a reasonably efficient algorithm for finding the discrete logarithm. If we use other underlying groups instead, we are able to prevent this. Other groups may allow other fast calculations, so thorough research is needed before we could safely use these groups. One important candidate to replace the multiplicative group is the family of elliptic curves. Though they combine good security with efficient calculations, it is hard to find appropriate elliptic curves. The elliptic curves are used by amongst others the NSA. Another interesting group is the family of braid groups. At the beginning of this century much research has been done is this field, which revealed a number of possible attacks. Since then the focus has switched to other non-abelian groups with a difficult conjugacy search problem. New discoveries in this field may find that the braid groups are useful or not. Until then it is not advised to use them for important secrets.

Chapter 2

Integer Factoring using Pollard's p-1 and Rho method

Written by *António Sousa and Roel Wijgers.*

This chapter focusses on the easy understandable integer-factorisation problem: Given n, find numbers p and q, such that $n = p \cdot q$. This paper focusses on some of Pollard's methods, namely the *p-1* method and the *rho* method. The term *smoothness* (r is *B-smooth* if $r = \Pi p_j$, with all p_j prime and $\leq B$) is a key issue for the *p-1* method. Given smoothness B, we can estimate the runtime of the *p-1* method with $O(B\log B(\log n)^2)$. The *rho* method runs in time $O(\sqrt{q}\ )$.

2.1 Introduction

Integer Factoring is a very relevant issue in the field of Cryptography. The factoring problem comes down to this: given n, find numbers p and q, such that $n = p \cdot q$. In principle it is not necessary for p and q to be prime, within the field of Cryptography it is however. Why it is so important that p and q are prime has to do with the run time complexity of factorisation methods, when p and q would not be prime, usually it would be much easier to find the factorisation of n. From now on we will assume that p and q are prime.

The factorisation problem has been extensively studied throughout the past decades, which has resulted in a number of algorithms solving certain instances of the problem. At present there still has not been found a general solution to solve the factorisation problem for random n and most probably we will never find such a general solution (unless P=NP, which is very unlikely).

A company called RSA-security makes use of the difficulty of finding the factorisation for large numbers (200-digit+) in their decryption and encryption methods. They work with a public and private key-system, where the private key is a pair (n, d), and the public key is a pair (n, e). All calculations will be done mod n and the encryption function is as follows: $enc(x) = x^e$ mod n, while the decryption function is: $dec(y) = y^d$ mod n . This works because the public and private key sort of cancel each other out so that $x^{d \times e}$ mod $n = x$. In the preliminaries this will be shown.

Below we give an example of an algorithm that is a general solution to the factorisation problem! Too bad we do not have unlimited computing time.

2.1.1 Trial-division method

The trial-division method is an easy understandable method, which most people would use to solve the factorisation problem. This method comes down to this: Enumerate a list L of prime numbers {2, 3, 5, 7, ...}. Given a certain challenge n, for all elements $l \in L$ simply calculate $rem = n\%l$. When the remainder value is zero, you have found one of the factors, the other factor would be n/rem. This method can be used to factor small numbers. As soon as n starts getting 20 bits or more, this method will not work anymore, because you would need too much computing power then.

2.1.2 Overview of this chapter

In the following section we will give some background information that is required to understand the facorisation algorithms that will be discussed in this chapter. Section 2.3 contains an overview of the $p - 1$ method, while section 2.4 contains an overview of the *rho* method. Eventually in section 2.5 we compare the performance of both methods.

2.2 Preliminaries

2.2.1 Number theory

Within the field of cryptography most calculations are done with modular arithmetics. This means we calculate modulo a certain fixed number, which is called the *modulus*. Within modular arithmetics you can define so called remainder classes.

Definition 2.1 *The remainder class of* a (*modulo* m) *is the set* $\bar{a} = \{ \ b \mid a \text{ - } b \text{ mod } m = 0 \ \}$. *The set of remainder classes is called* $\mathbb{Z}_m$.

Some example might give some insight in this definition. Suppose $a = 5$ and $m = 11$. When you look at the set $\bar{a}$, some elements from this (infinite) set are: { 5, -6, 16, 27, 115, ... }. Another definition tells us you can do calculations as normal within the set $\mathbb{Z}_m$.

Definition 2.2 *On the set* $\mathbb{Z}_m$ *addition, substraction and multiplication are defined as follows:* $\bar{a} + \bar{b} = \overline{a + b}$, $\bar{a} - \bar{b} = \overline{a - b}$, $\bar{a} \cdot \bar{b} = \overline{a \cdot b}$.

A nice illustration of these calculations is seen in the following example. Suppose we have two sets $\bar{a} = \{ 5, 16, 27, ... \}$ and $\bar{b} = \{ 2, 13, 24, ... \}$ (again $m = 11$), when we look at $\bar{a} \cdot \bar{b}$, we can choose from infinitely many options. Take for example $5 \cdot 2 = 10$, but also $16 \cdot 2 = 32 = 10 + 11 \cdot 2$ and $16 \cdot 13 = 208 = 10 + 11 \cdot 18$. According to the above definition all the calculations should end up in the same remainder class $\overline{a \cdot b}$, which is indeed the case. From now on we will let a remainder class be represented by it's lowest non-negative element (e.g. $\bar{a} = 5$).

Definition 2.3 *The multiplicative group modulo m, with notation $\mathbb{Z}_m^*$, is defined by: $\mathbb{Z}_m^* = \{ a | a \in \mathbb{Z}_m, \exists a' : a \cdot a' = 1 \}$.*

The above definition defined a group $\mathbb{Z}_m^*$ of remainder classes that have a multiplicative inverse modulo m. An example of an element of $\mathbb{Z}_m^*$ is $\bar{a} = 2$. This remainder class has a multiplicative inverse, namely $\overline{a'} = 6$, when we calculate $2 \cdot 6 = 1$, indeed the outcome is 1.

You might wonder what the size of this set $\mathbb{Z}_m^*$ is. There exists a function to calculate the size. This function is called the Euler-function $\phi(m) = |\mathbb{Z}_m^*|$.

Theorem 2.4 *$a \in \mathbb{Z}_m^*$ if and only if $gcd(a, m) = 1$.*

It is beyond the scope of this paper to prove that the above theorem is indeed valid. For those who are interested, we refer to [TEL-02]. The above stated theorem has some nice consequences. When we look at a prime value p and want to say something about the size of $\mathbb{Z}_p^*$, thanks to Theorem 2.4 we know that the size is equal to $p - 1$, because for all values $a \in \{ 1 .. p - 1 \}$ it is the case that $gcd(a, p) = 1$.

Theorem 2.5 *(Lagrange) For all $b \in \mathbb{Z}_m^*$ it is the case that $b^{\phi(m)} = 1$.*

Two very illustrative proofs of 2.5 can be found in [Tel02].

We end this subsection with a theorem that is the most important one to remember for this chapter.

Theorem 2.6 *(Fermat) For p prime and $a \in \mathbb{Z}_p^*$ it is the case that $a^{p-1} = 1$.*

Proof. Since p is prime, we know that $\phi(p) = p - 1$. So this theorem follows directly from Theorem 2.5. △

2.2.2 Number theory and RSA-security

In the introduction section we showed the method which the company RSA-security uses to encrypt and decrypt messages. When you want to create a public and private key pair, the first thing you do is choosing two large prime numbers p and q, and you calculate $n = p \cdot q$. When we look at theorem 2.4, we have all the knowledge required to calculate the size of $\mathbb{Z}_n^*$. There are in total $p - 1$ values that share a factor q with n, as well as there are $q - 1$ values that share a factor p with n. The value zero is not an element of $\mathbb{Z}_n^*$ as well. The size of $\mathbb{Z}_n^*$ is therefore $\phi(n) = n - p - q + 1$. When you calculated the size of $\mathbb{Z}_n^*$, you can start choosing a public key $e \in \mathbb{Z}_n^*$ which can simply be a random value from the set $\{1 .. \phi(n) \}$. Your private key will be the value $d = e^{-1} \bmod \phi(n)$. You can now forget about the factors of n and use your public and private key pair, such that $enc(x) = x^e \bmod n$ and $dec(y) = y^d \bmod n$.

2.2.3 Smoothness of numbers

The smoothness of numbers is a very important aspect with respect to Pollard's $p-1$ method. Before we will define what smoothness exactly is, there is one very important theorem from number theory that has not been mentioned yet.

Theorem 2.7 ***(Main theorem of arithmetics)*** *Any positive number n can be written as a multiplication of prime numbers, i.e. $n = \Pi_{i=1}^{k} p_i$. Apart from switching the order, this representation is unique.*

The above theorem does not state a prime number may occur only once in the composition of a certain number n, so this theorem is still valid for values like $40 = 2^3 \cdot 5$.

The term smoothness is defined as follows [Len00]:

Definition 2.8 *A number n is called B-smooth if all its prime factors are smaller than or equal to B.*

An important thing to note with respect to smoothness is how we should interpret the term prime factors. When we look at the value $40 = 2^3 \cdot 5$ again, we mean by prime factor the values 2^3 and 5, so with prime factor we mean a value p^k, with p prime and $k \geq 1$.

The value 40 is 8-smooth, as well as 100-smooth and 1000-smooth. However, 40 is not 7-smooth, since $2^3 > 7$.

2.2.4 Greatest common divisor

Below you will find an algorithm which computes the greatest common divisor of two numbers that is over 2300 years old. This algorithm has a worst case runtime of $O(n^2)$, where n denotes the highest number of digits of the input values a and b. This runtime complexity follows from the fact that in the worst case, there will be n recursive calls of the *gcd*-method. To calculate the mod of two numbers, one division operation has to be computed, which has a worst case runtime of $O(n^2)$, if n is greater than the natural size of the computer's arithmetic operations.

Algorithm 1 Euclides' *gcd*(a, b)

```
if $b = 0$ then
    return $a$
else
    $d \longleftarrow gcd(b, a\%b)$
    return $d$
end if
```

Casus 2.1: MODULE JAVA.MATH.BIGINTEGER

The BigInteger class allows us to do a lot of arithmetic operations with numbers of arbitrary size. The creation of these huge numbers goes like this:
BigInteger example = new BigInteger("12345678987654321");
BigInteger two = BigInteger.TWO;
Some examples of operations on BigIntegers are below:

$$\begin{aligned} z = x + y &\ :\ z = x.\mathsf{add}(y) \\ x == y &\ :\ x.\mathsf{compareTo}(y) == 0 \\ x \text{ is prime?} &\ :\ x.\mathsf{isProbablePrime}(1000) \\ z = \mathsf{gcd}(x, y) &\ :\ z = \mathsf{x.gcd}(y) \end{aligned}$$

2.2.5 Implementation of algorithms

We have implemented both of the algorithms in java, since we have a lot of experience with that programming language. If you want your algorithm to be implemented as efficient as possible, you would probably choose for a language like C or C++, but since we both used java for the implementations, we both have the loss of efficiency and we can still compare both methods quite good.

Casus 2.1 contains information about a very relevant module when you want to do calculations with numbers that exceed the size of normal integers or longs.

2.3 Pollard's *p-1* method

John Pollard published his *p-1* method in the year 1974 [Pol74]. As was mentioned earlier, this method relies on the concept smoothness. When you want to factor a number $n = p \cdot q$, we first assume that p is the smallest factor (which can be done without loss of generality) and we hope that the value $p - 1$ is B-smooth for a not too large value of B. The *p-1* method will be explained with pseudocode at first, but later on we will show how it looks in the java implementation.

2.3.1 Explanation and correctness of the algorithm

Probably you do not see yet that the *p-1* algorithm is indeed capable of finding a factor (sometimes). Therefore we will explain what is going on in this algorithm. You might wonder why you should assume a certain smoothness B, the idea behind this is the fact that we hope that the value $p - 1$ (where p is the smallest factor of n) is B-smooth. Suppose this is indeed the case, so $p - 1$ is indeed B-smooth. We can write $p - 1$ as $\Pi_{i=1}^{k} p^{r_i}$, according to theorem 2.7. We will slightly rewrite this to term to $\Pi_{i=1}^{k} P_i$, where P_i is a prime factor, i.e. $P_i = p_i^{r_i}$ for some value of r_i.

Algorithm 2 Pollard's *pMinOne*(n)

```
1: Assume smoothness $B$
2: $a \longleftarrow$ pick random value from $\{2 \,..\, n-1\}$
3: $a \longleftarrow a^{B!}$
4: $a \longleftarrow a - 1$
5: $d \longleftarrow \gcd(a, n)$
6: if $d$ unequal to 1 and unequal to $n$ then
7:     found factor $d$
8: else
9:     failed to find factor
10: end if
```

Let $\mathcal{P}$ denote the set of prime factors where $p-1$ consists of.

When we look at the value $B!$ we can rewrite this term with respect to $\mathcal{P}$:

$$B! = \Pi_{(f \in \mathcal{P})} f \cdot \Pi_{\substack{b \notin \mathcal{P} \\ b \leq B}} b$$

When we compute a value $a^{B!}$ this can be rewritten into:

$$a^{B!} = \left(a^{(\Pi_{(f \in \mathcal{P})} f)}\right)^{\Pi_{\substack{b \notin \mathcal{P} \\ b \leq B}} a^b}$$

Since we assumed that the value $p-1$ was B-smooth, with all it's prime factors in $\mathcal{P}$, we can further rewrite $a^{B!}$:

$$a^{B!} = \left(a^{p-1}\right)^{\Pi_{\substack{b \notin \mathcal{P} \\ b \leq B}} a^b}$$

Now we finally arrived at some interesting point. Although all the calculations in the $p-1$ algorithm are done modulus n, we know that $a^{p-1} = 1 \bmod p$. So when you would like to say something about the value $a^{B!} \bmod p$ some very interesting things happen. Because of Fermat's theorem 2.6, we know that $a^{p-1} \bmod p = 1$. So this means when we would look to the value $a^{B!} \bmod p$:

$$a^{B!} = 1^{\Pi_{\substack{b \notin \mathcal{P} \\ b \leq B}} a^b} = 1$$

When you substract 1 from $a^{B!}$, this means that $a^{B!} - 1 \bmod p = 0$, meaning that $a^{B!} - 1 \bmod n$ is a multiple of p. So when you compute the greatest common divisor of n and $a^{B!} - 1 \bmod n$, you will find the factor p since $p-1$ was B-smooth.

2.3.2 Implementation of the p-1 method

Like we have said earlier, we implemented the $p-1$ algorithm in java. In algorithm 3 we give the source code of how we implemented the method. Perhaps a bit suprising is the choice of $a = 2$ at the start, instead of choosing a random value from the set $\{$ 2, .. $n-1$ $\}$. This has two reasons:

- Since it does not matter which value you pick for a, the first few iterations of the for-loop can be computed faster, because the numbers are much smaller (2^k is much smaller than the expected $(\frac{n}{2})^k$ with a random a and k a small value). This statement is true untill the numbers start getting larger then n, after which $a = 2$ and $a = random$ will need about the same computational power.

- Theorem 2.4 requires that for $a \in \mathbb{Z}_m^*$, $\gcd(a, m) = 1$. It is always the case that $2 \in \mathbb{Z}_m^*$, for $m \geq 3$ and m a composite number of 2 primes, since m will be an odd number. Although the chances are small when you pick a random a you will find a value for which $a \notin \mathbb{Z}_m^*$, it is still better to know 100% sure that $a \in \mathbb{Z}_m^*$.

Algorithm 3 public BigInteger factorPminOne(**BigInteger** n, **int** smoothness)

```
{
    BigInteger a = new BigInteger( "2" ) ;
    for( int t=0; t < smoothness; t++ ){
        a = a.modPow( new BigInteger(""+t), new BigInteger(""+n) );
    }
    a = a.add( BigInteger.ONE.negate() );
    BigInteger d = n.gcd( a );
    boolean test1 = d.compareTo( BigInteger.ONE ) == 0;
    boolean test2 = d.compareTo( n ) == 0;
    BigInteger result = BigInteger.ONE.negate();
    if( !test1 && !test2 )
        result = d;
    return result;
}
```

2.3.3 Run-time complexity of the p-1 algorithm

When you look at algorithm 3, you can see there is one calculation which dominates the run-time complexity. This dominant factor is the for-loop, where we calculate $a^{B!}$. The body of the for-loop will be executed B times (where B denotes the smoothness parameter). To estimate the worst case run time of a single iteration in the body of the for-loop, we need to know how much time it costs to calculate an exponentation. An exponentation like $x^r \bmod n$ can be computed by repeated squaring and multiplication (for details how to do this, we refer to [Len00]). The number of times you need to square and multiply is bounded by $2{\cdot}\log_2 r$. The next thing we need to know is how much time a square or multiply operation costs. The worst case operation you can get is calculating $(n-1) \cdot (n-1)$. Now how much time does this cost? We use a very smart algorithm that makes use of the bitrepresentation of numbers:

$$a \cdot b = \begin{cases} (2a) \cdot (\frac{b}{2}) & \text{if } b \text{ is even} \\ a \cdot (b-1) + a & \text{otherwise} \end{cases}$$

We assume b is the largest value. This method takes constant time for the first case (testing 1 bit and moving one bit for a and b) and the second case will cost time proportionate to the size of b (i.e. $O(\log b)$, since we can sum bitwise). The number of recursive calls of this method is bounded by the value $2{\cdot}\log_2 b$. When we would look to the worst case multiplication $(n-1)\cdot(n-1)$, we see that the worst case runtime is $O((\log n)^2)$.

Now we can finally estimate how much time it costs to calculate the worst case factor $(n-1)^B \bmod n$, namely the number of times we need to square and multiply, times the complexity of one single multiplication. We find a complexity of $O(\log B(\log n)^2)$. Since the for-loop will be executed B times, we find a total complexity of $O(B\log B(\log n)^2)$.

2.4 Pollard's rho method

The Pollard's Rho Method (or Monte Carlo Factorization Method how it's also known), was published by John Pollard in 1974.

2.4.1 The algorithm

Pollard's method is based on the combination of various ideas.

The first one is the so-called *birthday paradox.*

The birthday paradox states that the probability of finding two persons with the same birthday exceeds 50% if we consider a group of 23 or more persons. Generalizing: picking numbers at random out of a set of p, the probability of picking the same number twice exceeds 50% after we pick $1.177\sqrt{p}$ numbers. So we can expect the first duplicate after $c\sqrt{p}$ for some small constant c

We're not going to prove the birthday paradox since there's already a lot of literature about it. This per se, doesn't help very much in factoring but let's keep this result in mind for later.

The second idea has to do with some proprieties of divisors of numbers and states that given p, a factor of n, and two random numbers a and b, such that $0 \le a, b \le n$, we can check that $a \equiv b \bmod p$ – i.e., a is congruent to b modulo p –, if $\gcd(|a-b|, n) > 1$.

We can easily see this is true because, if $a \equiv b \bmod p$, it means, by definition of modulus, that p divides $(a-b)$. Since p is a factor, it obviously divides n, so we have that p divides both n and $(a-b)$. If we find a $d = \gcd(|a-b|, n)$ then d, for sure, also divides n and $(a-b)$ (by definition of gcd). We can, then, conclude that $a \equiv b \bmod p$ (because of what we stated before) and, even better, if $1 < d < n$ we can also conclude that d is a factor of n, and so we have a factorization.

Combining this two ideas we know:

- a way to check when $a \equiv b \bmod p$
- that when it happens we may have a factorization
- that it will happen with a more or less nice probability (due to the birthday paradox)

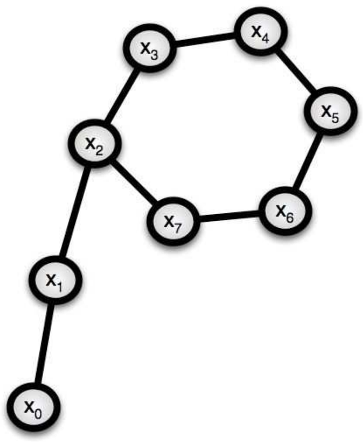


Figure 2.2: Random sequence mod p

So, we can build a factorization algorithm!

All we need to do is generate a sequence of random numbers $< n$ and compute the gcd. Instead of computing numbers at random we can generate a random sequence using a polynomial like $f(x) = x^2 + 1$ and compute it mod n. For this, we take a random x_0 and calculate $x_{i+1} = x_i^2 + 1 \bmod n$ as our random sequence. Since $x_i \bmod p$ behave more or less like random integers in $0, 1, ..., p - 1$ we can expect to find $x_i \equiv x_j \bmod p$ $(i \neq j)$, after $c\sqrt{p}$ elements have been computed (birthday paradox again!), which means also that we find a factorization at that time. However, this is not as simple as it looks. We have to consider approximately $(c\sqrt{p})^2/2$ pairs *and* to compute their gcd before we get to the value we want.

Luckily, we also have a way to make this better based on some properties of our random sequence. If we look at the fact that when $x_i \equiv x_j \bmod p$, then also $f(x_i) \equiv f(x_j) \bmod p$, it's easy to see that at some point our sequence (computed mod p) enters a loop. Actually, since we are computing the sequence mod p, we can only have p different numbers so it must always come to a repeated value at some point! That point is the one we are looking for, when the sequence folds on itself. Our sequence then goes on for a while and returns to some previous value (usually not the first one), biting it's own tail and entering a loop, like Figure 2.2. Is this looping that gives the method is name because the graphical representation of the sequence looks like the greek letter Rho (ρ).

But how can this help us? If we consider that the loop is of size t and that we're on position x_k where k is on the loop part of the sequence (not in the tail) and k is a multiple of t, then $2k$ is also a multiple of t. This means that $x_k \equiv x_{2k} \bmod p$ because it is the same point in the loop (since both are multiples of t). So, instead of computing the gcd for all values, we just run two sequences where one goes twice faster, i.e., $x_{i+1} = f(x_i)$ and $y_{i+1} = f(f(y_i))$. Then we compute $\gcd(|x_i - y_i|, n)$ until we get to the point we want. This way we can expect a factorization approximately after $2\sqrt{p}$.

Combining all this together, we have the algorithm for Pollard's Rho method in Algortihm 4.

Algorithm 4 Pollard's rho algorithm

```
1: $x \leftarrow 2, y \leftarrow 2, d \leftarrow 1$
2: if $n$ is prime then
3:     $n$ is factor
4: else
5:     while $d = 1$ do
6:         $x \leftarrow f(x)$
7:         $y \leftarrow f(f(y))$
8:         $d \leftarrow \gcd(|x - y|, n)$
9:         if $1 < d < n$ then
10:            $n' \leftarrow n/d$
11:            $fac(d)$
12:            $fac(n')$
13:        end if
14:        if $d = n$ then
15:            fail
16:        end if
17:    end while
18: end if
```

2.4.2 *Example*

Let's prove that it really works with a small example. Suppose we want to factor $n = 1717 = 17 * 101$ and we're using $f(x) = x^2 + 1$ as the random sequence generator. We take $x_0 = 2$ and $y_0 = 2$.

First we check if n is prime. Obviously it's not so we continue to the else statement and the algorithm goes on for k iterations as follows in table 2.1.

We can see that we find a gcd$>$ 1 when $k = 6$. We then try to try to factor that value and the result of it's division with n. So we do fac(17) and fac(101). Once again we check the primality of those numbers. Since they both are prime we have a factorization!

Now we can check that our previous assumptions are correct. As it's shown in table 2.2, when $k = 6$ we have $x_k \equiv y_k \bmod p$. We can also see that the random function has

Table 2.1: Factoring 1717

k	x_k	y_k	$\lvert x_k - y_k \rvert$	$\gcd(\lvert x_k - y_k \rvert, n)$
0	2	2		1
1	5	26	21	1
2	26	1608	1582	1
3	677	1600	923	1
4	1608	400	1208	1
5	1580	1098	482	1
6	1600	1328	272	17

Table 2.2: Factoring 1717 (2)

k	x_k mod 17	y_k mod 17
1	5	9
2	9	10
3	14	2
4	10	9
5	16	10
6	2	2

a loop.

2.4.3 Implementation

Since the method is quite simple, the implementation was straightforward. With the help of the `BigInteger` class, coding is almost a direct translation of the algorithm.

There is not much to say about the code. We can, however, make some considerations about the *BigInteger* class methods.

The primality testing offered by the class takes a integer as a parameter. That parameter is the value of *certainty*. Java API describes this value as the measure of uncertainty the caller is willing to tolerate (having in consideration the execution time, of course). This means that if this method returns true, then the probability of the number being prime is greater than $1 - \frac{1}{2^c}$ where c is the certainty value. The execution time of this method is proportional to the value of this parameter.

Nothing is referred about the `gcd` method but it's quite obvious that it's execution time is dependent of the number size.

The `f()` method corresponds to the random number function generator.

```
private void fac(BigInteger n) throws Exception {
  BigInteger x = new BigInteger("2");
  BigInteger y = new BigInteger("2");
  BigInteger d = new BigInteger("1");

  if (n.isProbablePrime(1000)) {
    System.out.println("Factor found: " + n.toString());
    } else {
      while (d.compareTo(BigInteger.ONE) == 0) {
        x = f(x, n);
        y = f(f(y, n), n);
        d = n.gcd((x.subtract(y)).abs());
        it++;
      }
      if (d.compareTo(BigInteger.ONE) == 1
          && d.compareTo(n) == -1) {
        BigInteger nn = n.divide(d);
        fac_aux(d);
        fac_aux(nn);
      }
      if (d.compareTo(n) == 0) {
        throw new Exception("Failure: d = n");
      }
    }
  }
}
```

Despite the fact that in our implementation x_0, y_0 and $f(x)$ are fixed, it's possible to use different values in the method. It can even improve the performance of it.

2.4.4 Some tests and performance considerations

When testing the algorithm, we can't use as measure the execution time since that is dependent both on the *hardware* and on the Java implementation. So we used the number of iterations. Like we said in the previous sections, the expected number of iterations is $2\sqrt{p}$ with p being the smallest factor. The algorithm behaves nicely staying way under $2\sqrt{p}$ iterations most of times (way way less if we're lucky).

Not entering into very detailed statistics, from our tests we can conclude that the algorithm factors numbers with p up to 10^{12} in less than 1 minute. Between 10^{13} and 10^{14} in a couple of minutes; for 10^{15} it starts to take a little longer, up to 15 minutes. If $p > 10^{16}$ we can start to count the time in hours. It goes from half hour to 2 hours. More than that we didn't try (too long time waiting!).

However, we could estimate the average time of an iteration as being of 25ms. So, we can estimate to find a factor in approximately $25 \cdot 2 \cdot \sqrt{10^k} = 50 \cdot 10^{\frac{k}{2}}$, with k being the number of digits of the smallest factor (p). As you may realize, it starts to take really long time to factor numbers with more than 30 digits (even centuries!).

One way of increasing performance could be, for instance, to run z pairs of sequences, each one in it's own thread and with different values for x_0, y_0 and $f(x)$. This way the probability of finding the wanted value would be reduced by z. Of course, because we are using threads, we need to make a compromise about z, otherwise, instead of increasing we'll get a degradation of performance.

2.5 Conclusions and results

Since we implemented both the *p-1* and *rho* algorithms, we found out what kind of composite numbers can be factored with these methods. Both methods run very well if one of the factors is small (i.e. 15-digits), when this is the case, both methods can easily compute the factors of 500-digit numbers! One major drawback of the $p - 1$ method is the fact that you need to know the smoothness in advance, for the method to work. When you look at the *rho* method you do not need any parameter and this makes it far more preferable than the $p - 1$ method. If you want to make a good comparison in runtime of the two methods, you would need to do some excessive testing, what would require a more thorough study of the subject.

We did some research on the results of the methods and found out that one of the greatest factors ever found with the $p-1$ algorithm was a 58-digit number (found by P. Zimmermann):

$$p = 1372098406910139347411473978297737029649599583843164650153,$$

$$p-1 = 23\cdot 32\cdot 1049\cdot 1627\cdot 139999\cdot 1284223\cdot 7475317\cdot 341342347\cdot 2456044907\cdot 9909876848747$$

which is a factor of $2^{2098}+1$.

It would be unfeasible to try to find this factor with the *rho* method, since you would require far too many computing power (in the order of 10^{29}).

For the Pollard's rho method, the most remarkable success was the discovery, in 1980 by Pollard and Brent, of the factorization of the eight Fermat number:

$$2^{2^8}+1 = 1238926361552897 \cdot p62$$

where $p62$ denotes a 62-digit prime number.

The final conclusion we make here is the fact that the $p-1$ algorithm can factor much larger numbers than the *rho* method. You need to be very lucky though, since the probabilty that the value $p-1$ is B-smooth for huge p and $B \leq 10^{13}$ is very low.

Chapter 3

Threshold cryptography

Written by *Ron de Bruijn and Adriano Galati.*

Threshold cryptography enables groups of users to share a secret, such that a subset of the users is needed for doing some computation which requires the secret. Common applications include signing messages as a group and decrypting messages.

A threshold cryptosystem is a system in which there are N users or players who share a distributed secret. Typical systems have some kind of master key system. The disadvantage of these systems is that the person holding for example the Euler toitent $\phi(n)$ does not need any of the other users of the system to do the desired computation(and all the keys of the other participants are compromised). The system described in ([KY02]) does not have this disadvantage and will be discussed in the remainder of this paper.

3.1 Threshold cryptosystems

Theoretically speaking, threshold cryptography is an instance of generic multi-party computation, since every player in the system holds some secret information which it does not want to share with the rest, although the sharing of an end-result of certain computations based on that piece of secret information is admissable.

Now, one could say: problem solved. However, to make things *practical* threshold cryptosystems must become more efficient than the protocols used in generic multi-party computations. The disadvantage of the generic protocol is that it requires "secure circuit evaluation" which have a large communication overhead. It is also important that the specific method (i.e. decryption by a group or individual) of decryption is hidden from individuals who want to communicate to the group.

We will give a overview of [KY02], which is in fact targeted at making threshold cryptography *practical* and give some more details of extensions described in this paper.

In a k-out-of-ℓ threshold scheme at least k players must cooperate to complete the computation (e.g. the decryption of a message or the creation of a signature for a message). More over the protocol we will be discussing is *robust* and *proactive*. Both of these properties are described in section 3.1.1.

3.1.1 Properties of threshold cryptosystems

Just like a standard cryptographic scheme can be described by a number of properties, like e.g. having "perfect security" or only "computational security", threshold cryptosystems consider some more properties. In a k-out-of-ℓ threshold scheme there are these additional basic properties:

- *Robustness* refers to whether the system can withstand up to $k - 1$ faulty players, where $k \leq \ell/2$
- Proactiveness, time is sliced in to a number of time frames. A non-adaptive[1] adversary who can control up to $k - 1$ players in one time frame, but these not necessarily have to be the same in different time frames, still is unable to disrupt the computation or to do the computation wihout the help of the other players. In particular it is possible that the adversary gets control of every machine at some point in time. When the scheme can resist such adversaries, it is proactive. It is called proactive, because the system does not sit and wait until someone has broken the keys, but instead actively refreshes keys periodically.

3.1.2 A threshold homomorphic decryption scheme

Actually, at least the title of [KY02] ("Threshold Cryptosystems Based on Factoring") is a simplification, since it is based on both that factoring large numbers is hard and that the Quadratic Residuosity Assumption holds(see section 3.8). While the quadratic residuosity can be calculated by knowing the factors, whether the reverse is true is unknown.

[KY02] start out by creating a scheme which enables them to decrypt messages sent by the Goldwasser-Micali(GM) cryptosystem. Since without an understanding of the GM system it is pointless to discuss how to distributively decrypt messages, a short discussion of this system is included next.

3.1.3 The Goldwasser-Micali(GM) cryptosystem

The GM system is interesting in more than one sense. For one, it was the first to introduce the concept of Semantic Security, which was later used in the ElGamal system a.o. Also, it took a fundamental step to just solve the problem of encrypting one bit safely. While other systems had to make assumptions about the message space and probability distribution to be proven secure, their system was safe for all message spaces. It is for example insecure to encrypt messages in a very small message space with RSA(since it lacks Semantic Security). The GM key generation is described in Protocol 3.1, encryption in 3.2 and finally decryption in 3.3. Note that $\mathbb{Z}_n^1 = \{x | x \in \mathbb{Z}_n^* \wedge (x/n) = 1\}$. Remember that (x/n) is the Jacobi symbol(see 3.8.2).

[1]Adaptive adversaries can be handled through extensions to the protocol.

1. Randomly select two k-bit primes p_1 and p_2.
2. Set $n = p_1 p_2$
3. pick $y \in \mathbb{Z}_n^1$ such that y is a quadratic nonresidue modulo n
4. output (n, y) as the public key, and (p_1, p_2) as the private key for Alice, respectively.

Protocol 3.1: GM KEY GENERATION

3.1.4 A threshold homomorphic decryption scheme(continued)

With the GM machinery in place we can now continue with the real distributed decryption of a message sent to the group. Actually, the scheme used in [KY02] differs from [GM84] since they require N to be a Blum (see 3.8.4) integer. The reason for using a Blum integer is that if [SCP98] N is a Blum integer, then $(-1) \bmod x$ is a quadratic nonresidue with Jacobi symbol 1. This implies that on input a Blum integer N, it is easy to generate a random quadratic nonresidue in $\mathbb{Z}_N^*$: randomly select $r \in \mathbb{Z}_N^*$ and output $-r^2 \bmod N$. See Casus 3.4 for an example. See finally, if x is a Blum integer, given its prime factors p, q, it is possible to compute square roots modulo N in deterministic polynomial time. This last property is similar to the property stated in 3.1. This generation of an encryption fits indeed in the scheme of [GM84].

To decrypt first the Jacobi symbol $J = (C/N)$ is calculated. When $J \neq 1$, then the encryption failed, otherwise we calculate $b\prime = C^{(N-p-q+1)/4} \bmod N$. This formula can be derived from the Euler criterion, and a property of congruences. If $b\prime = 1$ then $b\prime$ is a quadratic residue. The original bit b can be recovered as $b = (1 - b\prime)/2$.

The above described decyption can also be done distributively. See 3.5 for the dealing protocol, and 3.6 for the decryption. In this case we have a veto-scheme, since all of the ℓ players are required for decryption of a message. When a veto-scheme is undesirable [FGMY97] can be used to make a k-out-of-ℓ scheme, although this method depends for security on the random oracle model (see 3.8.5). More interesting is the approach of [Rab98] who provide a *generic* method for converting a ℓ-out-of-ℓ scheme to a k-out-of-ℓ scheme.

Lemma 3.1 *Given the prime factorization of a composite integer n, deciding whether $q \in \mathbb{Z}_n^*$ is a quadratic residue* mod$n$, *can be done in $O(|n|^3)$ time.*

Now, assume Bob wants to send Alice the message $b = b_1 \cdots b_l$, where each b_i is one bit.

```
for each $b_i \in b$
        Bob picks $x \in \mathbb{Z}_n^*$ at random
        if $b_i = 1$ Bob sets $e_i = yx^2 \bmod n$
        else Bob sets $e_i = x^2 \bmod n$
Bob sends Alice $(e_1, \cdots, e_k) = E_n(b)$.
```

Encoding an l-bit message b takes $O(lk^2)$ time, where k is a security parameter. In general, one bit of cleartext is expanded into k bits of cyphertext.

Protocol 3.2: GM ENCRYPTION

Assume Alice receives $(e_1, \cdots, e_l)$ the encryption of a message b. Then,
for each $e_i \in e$,
Alice sets $b_i = 1$, if e_i is a quadratic residue, and 0 otherwise.
Alice sets $b = b_1 \cdots b_l$.
Computing b, from its encryption requires $O(lk^3)$ time.

Note that only Alice can compute the quadratic residuosity since she knows the factors of n (assuming QRA and the assumption that factoring is hard).

Protocol 3.3: GM DECRYPTION

Since the GM system is based on the QRA and based on the above lemma, it is undesirable to have a threshold system where the factors of n are known to a single party, since it basically defeats the point of needing at least k players to decrypt a message. Therefor, the modulus N has to be generated in a distributed manner.

3.2 A Simplified Approach to Secure Threshold and Proactive RSA

The issue of maintaining secret signing keys in a distributed fashion for long periods of time, while enabling uninterrupted signing capabilities is addressed by threshold signatures and proactive security.

Threshold signature schemes allow a group of players to "hold" the key and enable

Casus 3.4: EXAMPLE OF CREATING NONQUADRATIC RESIDUES

We need a Blum integer. 21 is a good choice, since $7 \cdot 3 = 21 \wedge 7 = 3 = 3 \bmod 4$. Now, for an example: suppose we pick as a random number 5 ($\in \mathbb{Z}_N^*$). (5 is $\in \mathbb{Z}_N^*$, since $5 \cdot 17 = 1$)
We get:

$$
\begin{aligned}
-5^2 \bmod 21 &= -25 \bmod 21 && (3.1)\\
-25 \bmod 21 &= 17 \bmod 21 && (3.2)\\
17 \bmod 21 &= 17 && (3.3)
\end{aligned}
$$

And, now 17 is not a quadratic residue, since there is no solution for $17 = x^2 \bmod N$. As a proof: these are all the squares in $\mathbb{Z}_{21}$:

$$1, 4, 9, 16, 4, 15, 7, 1, 18, 16, 16, 18, 1, 7, 15, 4, 16, 9, 4, 1, 0$$

As you can see 17 is not in this list, and therefor is not a square.

Dealing phase
Input: Composite N and primes p,q ($|p| = |q| = n$): $N = pq \wedge p, q = 3 \bmod 4$
Choose p_1, q_1,. . ., p_ℓ, q_ℓ at random from $(0, 2^{2n}) : \forall i : p_i = q_i = 0 \bmod 4$
Set $p_0 = p - \sum_{i=1}^{\ell} p_i$ and $q_0 = q - \sum_{i=1}^{\ell} q_i$
Send (p_i, q_i) to player i Broadcast (N, p_0, q_0)

Protocol 3.5: ℓ-OUT-OF-ℓ DEALING IN ADAPTED GM SCHEME

them, as a group, to produce signatures. We shall refer to such a scheme as a *(t; n)-threshold signature scheme* if there are n players and given a message m, any subset of players of size at least $t + 1$ can generate the signature on m.

Definition 3.2 *The scheme is t-secure if no subset of at most t players can compute a signature on a message not previously signed.*

Definition 3.3 *The scheme is t-robust if it can correctly compute signatures even in the presence of up to t corrupted players.*

In a distributed RSA, the signing key d (the RSA exponent) has to be maintained in an additive form, among n players $P_1, ..., P_n$. Player P_i would hold a value d_i, where $d = d_1 + \ldots + d_n$. This allows for a straight forward signing scheme, as $m^d = m^{d_1} \times ... \times m^{d_n}$. Thus, each player P_i gives the value m^{d_i} which is called the partial signature, and all the partial signatures are combined to form the complete signature on m.

The major drawback of the n-out-of–n additive solution is that it does not provide for threshold (and hence *robustness*); if each one of these shares is held by a different player, and one of the players crashes we have lost the secret key.

However we assume that this computation model is composed of a set of n *players* $P_1, ..., P_n$ who are connected by a complete network of private (i.e. untappable) point-to-point channels. In addition, the players have access to a dedicated broadcast channel; by dedicated we mean that if player P_i broadcasts a message, it will be recognized by the other players as coming from P_i. We make these assumptions (privacy of the communication channels and dedication of the broadcast channel) so that we can focus on a high-level description of the protocols.

The robustness is achieved by generating share back-up, i.e. each such additive-share will be further shared using a robust threshold verifiable secret sharing (VSS) scheme.

Decryption phase
Input: Ciphertext C
All players compute J=(C/N) If $J \neq 1$, all players output $\perp$ and stop.
Otherwise ($J = 1$), player i broadcasts $b_i = C^{(-p_i - q_i)/4} \bmod N$
All players publicly compute $b_0 = C^{(N - p_0 - q_0)/4} \bmod N$
The decrypted bit b is computed as $b = (1 - \prod_{i=0}^{\ell} b_i \bmod N)/2$

Protocol 3.6: ℓ-OUT-OF-ℓ DECRYPTION IN ADAPTED GM SCHEME

Input: secret key $d \in Z_{\phi(N)}$, composite N, element g of high order in Z_N^* number of players n and threshold t.

1. Choose and hand P_i value $d_i \in R[-nN^2...nN^2]$ for $1 \leq i \leq n$, set $d_{public} = d - \sum_{i=1}^{n} d_i$.
2. Compute and broadcast the witness $w_i \triangleq g^{d_i} \bmod N$, $1 \leq i \leq n$.
3. Share the value d_i using Feldman-Z_N-VSS Sharing Phase (Figure 3.9) on input, value d_i, high order element g, composite N , the number of players n, and threshold t.

Protocol 3.7: Secret key Distribution

Every time a signature has to be generated, each player will produce a partial signature under his additive-share so that all partial signatures are combined to generate the signature. First of all we will first verify whether this combined signature is valid, in order to optimize the protocol complexity, and then, if it is not, we will proceed to detect the players who have given incorrect partial signatures. Anyway, if a player provides an incorrect partial signature then his additive-share is reconstructed from the back-up. In order to make the composition of the additive-shares and the shares from back-up work properly, an underlying assumption has been made, and it is that the back-up value of a specific player is in fact the additive-share held by him.

Considering that the RSA key generation took place and the RSA key pair has been computed we have achieved the private key d and the public key (N, e), in which $N = pq$ and p, q are primes of the form $p = 2p' + 1, q = 2q' + 1$ and p', q' are primes, and where $ed \equiv 1 \bmod \phi(N)$.

Furthermore, the parameters of the system are set, i.e. the number of players n and the threshold t, where $n \geq 2t + 1$. An element g of high order is computed by setting $g \triangleq {g_0}^{L^2} \bmod N$ in order to enable the proof of security where g_0 is an element of high order and $L \triangleq n!$.

Afterward the dealer makes sharing the key d by generating shares and then backing-up each one of the shares using a VSS protocol.

At this point each players has to be able to verify partial signatures generated under each additive-key d_i. To do that, for each additive-key the dealer generates a witness signature in the form $g_{d^i} \bmod N$.

This witness will be utilized in the signature generation protocol in order to verify partial signatures. Furthermore, in order to unify the description for the players we added a public share d_{public}, such that $d = d_{public} + \sum_{i=1}^{n} d_i$. The steps carried out by the dealer are described in Figure 3.7.

Thus, given a message m, its signature under the global private key is $m^d \bmod N$ and it is achieved by the players in a distributed manner where each individual player uses his partial key, and we have that $m^d = m^{d_{public} + \sum_{i=1}^{n} d_i} = m^{d_{public}} \Pi_{i=1}^{n} m^{d_i} \bmod N$. In this way it is possible to compute the correct signature from the partial signature only if each player publicizes the correct value $m^{d_i} \bmod N$ (*partial signature*).

The protocol for signature generation is described in Protocol 3.8.

If an error is detected each player has to prove that he has generated his partial

Public information: high order element g, composite N w_i mod N for $1 \leq i \leq n$ witness for P_i's partial key.
Input: message m

1. Player P_i publishes partial signature $\sigma_i = m^{d_i}$ mod N.
2. P_i proves using the protocol of [GJKR96] that $DL_m\sigma_i = DL_g w_i$.
3. If proof fails for player P_i all players reconstruct d_i using the Feldman-Z_N-VSS Reconstruction Phase (Figure 3.9), and compute $\sigma_i = m^{d_i}$ mod N.
4. Set $SIG(m) = m^{d_{public}} \Pi_{i=1}^{n} \sigma_i$ mod N.

Protocol 3.8: Signature Generation

signature properly using a public witness w_i mod N which can be viewed as a commitment to the value d_i. This phase is carried out by the Verification Protocol described in depth in the section 3.6. If the partial signature is not proved, the player will retrieve his share from the back–up of the VSS protocol Reconstruction Phase. In fact, in VSS protocol a secret value $s \in [-nN^2...nN^2]$ is shared in a verifiable method. The protocol for signature generation is described in Figure 3.9.

Theorem 3.4 *The protocol described in Figure 3.9 is a verifiable secret sharing scheme satisfying the properties of unanimity, acceptance of good secrets, verifiability and unpredictability.*

Lemma 3.5 *Given a t-adversary who can corrupt at most t players, the view of the adversary of the secret shares generated by the protocol Feldman-Z_N-VSS of a secret s using a polynomial $f(x)$, such that $f(0) = s$, and of the sharing of a random secret r by a polynomial $r(x)$ with coefficients taken from the appropriate range are statistically indistinguishable.*

Lemma 3.6 *Assume elements g_0, $g \triangleq g_0^{L^2}$ $(L = n!)$ of maximal order in Z_N^*. Given values $\alpha_1, ..., \alpha_t$ and an additional value $g^s \bmod N$ it is possible to compute values $g^{a_1}, ..., g^{a_t} \bmod N$ such that the polynomial $f(x) \triangleq a_t x^t + ... + a_1 x + sL$ satisfies that $f(i) = \alpha_i$ for $1 \leq i \leq t$.*

Theorem 3.7 *Under the assumption that factoring is intractable the protocol described in Figure 3.7 and 3.8 is a constant round secure, robust, threshold RSA scheme, in the presence of t malicious faults where the total number of players is $n \geq 2t + 1$.*

We can notice the adversary who sees all the information of the corrupted players and the signature on m, could generate by itself all the other public information produced by the protocol.

Sharing Phase:
Input:
secret value $s \in [-nN^2...nN^2]$ and composite N
element g of high order in Z_N^*
n number of players, t threshold value

The dealer carries out the following steps:

1. Choose $a_t, ..., a_1 \in [-nL^2N^3...nL^2N^3]$ and define $f(x) = a_t x^t + ... + a_1 x + sL$.
2. Compute $f(i) \in Z$ for $1 \leq i \leq n$ and $b_t \triangleq g^{a_t}, ..., b_1 \triangleq g^{a_1}, b_0 \triangleq g^{sL} \bmod N$.
3. Hand player P_i the value $f(i)$ and broadcast $b_t, ..., b_0$.

Verification steps

1. Player P_i verifies that $g^{f(i)} = \Pi_{j=0}^{t}(b_j)^{i^j}$. If the equation is not satisfied he requests that the dealer make $f(i)$ public.
2. The dealer broadcasts all shares requested in the previous step, if he fails to do so he is disqualified.
3. Player P_i carries out the verification of *Step 1* for all public shares. If the verification fails the dealer is disqualified.

Reconstruction Phase:
Input: high order element g, composite N , number of players n, threshold t values $b_t, ..., b_0 \bmod N$.

1. Player P_i broadcasts $f(i)$.
2. P_i finds a set I of size $t+1$ of indices such that $\forall i \in I$ it holds thats that $g^{f(i)} = \Pi_{j=0}^{t}(b_j)^{i^j}$.
3. P_j chooses a prime $P > 2nN^2$ and computes the secret s, $s \triangleq \sum_{i \in I}^{j} f(i) \prod_{j \in I, j \neq i} \frac{-j}{i-j} / L \bmod P$.

Protocol 3.9: FELDMAN–Z_N–VSS

3.3 Proactive RSA

Proactive signature schemes use threshold signature schemes as the basis but drastically reduce the assumption concerning failures. The lifetime of the system is split into time periods, and in each such time period it is assumed that no more than t players will be corrupted. We shall say that a scheme is *t-proactive* if it is secure and robust even in the presence of up to t corrupted players in every time period, where the set of corrupted players can change arbitrarily from one time period to the next. Thus, an attacker wishing to compromise the system will need to corrupt $t + 1$ players in a short period of time.

Basically, *proactiveness* is achieved through changing the representation of the key, but not its value. The representation of the key will change from $d_1 + ... + d_n$ to $d_1^{new} + ... + d_n^{new}$, as the representation of the back-up. This is for two reasons:

1. the first being that it contains the current representation and thus it should be removed,

2. the second is that we need to have a valid back-up for the new representation, otherwise the backup would be worthless.

During the proactivization protocol execution each player takes his additive-share d_i of the key d and sum it up to d_i, then he gives each player such a sub-share. Afterwards each player adds up all the subshares which he received in order to achieve his new share for the new time period. At this point it is clear that the new sharing is still a sharing of the key d, even though the sharing is totally independent of the previous sharing. Of course, to produce the new share back–up the player uses the VSS protocol, while to ensure the correctness of the proactiveness he verifies the sum of the sub-shares is the value of the share d_i by using commitments or a witness of the form $w_i^{new} = g^{d_i^{new}} \bmod N$ to the new additive sub-shares and checks the back-up generated by using the VSS is the new additive share held by the player.

The share-refreshing protocol for the update period is presented in Figure 3.10.

Theorem 3.8 *Under the assumption that factoring is intractable, the protocol described in Figures 3.7, 3.8 and 3.10 is a constant round secure, robust, threshold proactive RSA scheme, in the presence of t malicious faults where the total number of players is $n \geq 2t + 1$.*

3.4 Robust and Efficient Sharing of RSA Functions

The addition of the verification protocols to a non-robust threshold signature scheme has an obvious performance cost. However, under normal circumstances when the system is not under active attack one can avoid this extra overhead.

The trade-offs between different properties of a fault-tolerant threshold RSA depend on the particular application of these techniques. For example, some applications may require minimal communication between the players during the Signature Phase. Other might need that partial signatures will be “publicly verifiable”, namely, that any person, even outside the group of N players, can verify every partial signature. In this paper we described two different protocols for verification of partial signatures to address these different requirements.

The first protocol has a *non-interactive* Signature Phase, and requires each player to hold local secret verification data. The second protocol requires interaction in order to verify partial signatures, even if all the verification data is public.

Public information: element g, composite N w_i mod N for $i \leq i \leq n$
Input of player P_i: secret share d_i such that $g^{d_i} = w_i$ mod N

1. Player P_i randomly chooses values $d_{i,j} \in_R [-N^2..N^2]$ for $1 \leq j \leq n$, sets $d_{i,public} = d_i - \sum_{j=1}^{n} d_{i,j}$, computes and broadcasts $g_{i,j} \overset{\triangle}{=} g^{d_{i,j}}$ mod N.

2. Player P_i sends to player P_j the value $d_{i,j}$.

3. Verification of distribution of proper share size and public commitments: P_j verifies that $d_{i,j} \in [-N^2..N^2]$ and that $g_{i,j} = g^{d_{i,j}}$ mod N if not then he requests that $d_{i,j}$ be made public and set $g_{i,j}$ to g raised to this public value.

4. If P_i does not cooperate in Step 3 then his value d_i is reconstructed using the Reconstruction Phase (Figure 3.9).

5. Verification that the sub shares in fact sum up to the previous share of P_i: P_j verifies that $w_i = g_{i,public} \prod_{j=1}^{n} g_{i,j}$ mod N, if not then player P_i's share d_i is reconstructed using the Reconstruction Phase.

6. P_i computes his new share $d_i^{new} = \sum_{j=1}^{n} d_{j,i}$ (note that the value $g^{d_i^{new}}$ mod $N = \prod_{j=1}^{n} g^{d_{i,j}}$ mod N is already public), and shares it using Feldman-ZN-VSS Sharing Phase. This results in a value g^{sL} mod N where s is the secret that P_i shared.

7. If P_i fails to share his secret or $(g^{d_i^{new}})^L \neq g^{sL}$ mod N then each player P_j exposes $d_{j,i}$. If P_j fails to expose $d_{j,i}$ then d_j is reconstructed by all players.

Protocol 3.10: Share–refreshing Protocol

3.5 Non-Interactive Robust Threshold RSA

3.5.1 Generation of Verification Data (Dealing Phase)

The *non-interactive solution* considers two players P and V in which P generates a signature and V wants to verify it. The prover P holds the secret key d and y. The verifier V holds b and c, such that $y = bd + c$. The values d, y, b and c are dealt to the corresponding parties during the dealing phase (and kept secret by the parties). Given a message m, the prover generates the partial signature $m^d \mod n$, and the additional information $m^y \mod n$. P gives these values to V, who verifies the partial signature by checking whether $(m^d)^b m^c = m^y \mod n$.

To generate the data for verification of partial signatures the dealer simply runs the ICP-Gen-Integers (Information Checking Protocol) 3.11 for every pair of players P_i and P_j where P_i is the prover P, P_j is the verifier V, and d_i is the P_i's secret partial key. P_i holds the following values:

Input: RSA composite n, secret value $d \in [\phi(n)]$ known to D, security parameters $0 \leq k_1, k_2, k_3 = k_1 + k_2 + logn$.

1. Choose $b \in [2^{k_1}]$ and $c \in [2^{k_3}]$ with uniform distribution.
2. Compute $y = c + bd$ over the integers.
3. Secretly transmit d and y to the prover P.
4. Secretly transmit b and c to the verifier V .

Protocol 3.11: ICP-Generation Protocol

- His share d_i.
- Auxiliary authentication values $y_{i,1}, ..., y_{i,N}$ where $y_{i,j} \in Z$ is used to prove his partial signature to P_j.
- Verification data $V_{1,i}, ..., V_{N,i}$, where $V_{j,i} = (b_{j,i}, c_{j,i}), b_{j,i} \in [2^{k_1}] and c_{j,i} \in [2^{k_3}]$. For each j, the pair $V_{j,i}$ is used to verify the correctness of P_j's partial signatures.

3.5.2 Partial Signature Verification (Signature Phase)

It is very easy to understand the protocol for the partial signature verification just by looking the Figure 3.12. In the context of the *Signature Phase* this protocol will be carried out by every pair of players. After executing all these invocations of the *Non-interactive Verification Protocol*, player P_i will take a subset of $T + 1$ partial signatures which he has accepted, and will generate the signature on m.

Input: Player V: $b \in [2^{k_1}], c \in [2^{k_3}]$.
Player P: $d \in [\phi(n)], y = bd + c$.
Both players: message $m \in Z_n^*$, RSA composite n.

- P broadcasts the signature $\hat{S}_m = m^d \mod n$ and the auxiliary value $Y = m^y$ mod n
- V checks if $\hat{S_m^b} m^c = Y$. If yes, he concludes that $\hat{S_m} = \pm m^d \mod n$ and accepts it.

Protocol 3.12: The Non-Interactive Verification Protocol

Input:
Public: RSA modulus .
Dealer D: key-shares $d_i \in [\phi(n)], for i = 1, 2, ..., N$.
1. D chooses a random value w of high order in Z_n^* and broadcasts values $w_i = w^{d_i} \mod n$ for $i = 1, 2, ..., N$.

Protocol 3.13: The Sample Partial Signature Generation

3.6 Interactive Robust Threshold RSA

The basic idea underlying the interactive solution is that RSA signatures can be verified even if the verification exponent is unknown to the verifier.

3.6.1 Generation of Verification Data (Dealing Phase)

In this protocol the verification data dealt during the initialization protocol (and used during the signature phase to verify partial signatures) consists of a random public sample message w and and its corresponding sample partial signatures $w^{d_i} \mod n$, for each one of the partial keys d_i held by the players. The sample signatures are broadcast to all players, and can even be made public (Figure 3.13). The verification data is $V_{i,j} = w_j = w^{d_j} \mod n$, for all i, j. The first difference between interactive and non-interactive protocol is that here the $V_{i,j}$'s are public, and second one is that the commitments to the partial key also generate a commitment to the signing key d, i.e. the value $w^d \mod n$ is also made public.

3.6.2 Verification of Partial Signature (Signature Phase)

During the signature verification phase each partial signature can be verified by the other participants by interacting with the signer, namely each player P_i checks the partial signatures produced by each other player P_j. To be as clear as possible it is simple to consider only two players, the signer P and the verifier V. The player P has his secret signature key $d \in [\phi(n)]$ and both players have access to a publicly known sample message w and its partial signature $w^d \mod n$. For any $x \in Z_n^*$ we denote by S_x the corresponding signature of P on x under the key d, namely, $\hat{S}_x = x^d \mod n$. By $\hat{S}_x$ we denote a string claimed but not yet verified to be the signature of x.

Commitment functions can be implemented in many ways, in particular using a semantically secure encryption, i.e. by using commitment functions based on the RSA function where the private key is not known to V and the public key is known to both parties. Commitment to A is done by P by encrypting this value concatenated with a random string, while decommitment is done by having P reveal both A and the string r used for the encryption. This implementation of a commitment function is very efficient as it involves no long exponentiations.

To conclude we can infer that the combination of the secure threshold RSA scheme with the robustness techniques developed above yields two secure solutions to threshold RSA signature schemes.

Input: Prover: secret $d \in [\phi(n)]$.
Common: RSA composite n, sample message $w \in Z_n^*$, signature S_w, message $m \in Z_n^*$, claimed $\hat{S}_m$.

1. V chooses $i, j \in_R [n]$ and computes $Q \overset{def}{=} m^i w^j \mod n$ $V \rightarrow P : Q$
2. P computes $A \overset{def}{=} Q^d \mod n$ and *commit*(A) $P \rightarrow V : commit(A)$
3. $V \rightarrow P : i, j$
4. P verifies that $Q = m^i w^j \mod n$ If equality holds then P decommits to A
5. V verifies that $A = \hat{S}_m^i \hat{S}_w^j \mod n$. If equality holds then V accepts $\hat{S_m}$ as the signature on m, otherwise it rejects.

Protocol 3.14: Interactive Verification

3.7 Conclusion

Threshold cryptosystems provide new options for enhancing security. There are two main types of schemes, veto-schemes (in which the threshold is as large as the number of players) and threshold-schemes, in which only k-out-of-ℓ share holders are need to do a certain computation(e.g. decryption or signing of a message). Threshold schemes have been created which allow for modularly adding proactiveness, increasing/decreasing the theshold(k). There clearly has been a lot of research done for creating secure, communication efficient, indeed, practical systems. Despite this last sentence, we think that actually *implementing* these systems securely is a different matter on the simple basis that finding papers about the subject is easy, while finding software that can be used is a different story. In theory everything most systems do work "modularly", though. So, in order to make systems really "practical" it would help when there would be more than just a system on paper.

3.8 Appendix

Some number theoretic definitions are listed here.

3.8.1 Quadratic residue

If $\exists x : q \in \mathbb{Z}_n^* \wedge q = x^2 \bmod n$ then q is a quadratic residue mod$n$, otherwise it is a quadratic nonresidue mod$n$.

3.8.2 Jacobi symbol

The expression $(\frac{a}{n})$ in the following formula is called the Jacobi symbol.
$\left(\frac{q}{n}\right) = \left(\frac{q}{p_1}\right)^{\alpha_1} \left(\frac{q}{p_2}\right)^{\alpha_2} \cdots \left(\frac{q}{p_k}\right)^{\alpha_k}$

In the case where $\left(\frac{q}{n}\right) = 1$ it is hard to decide whether q is a quadratic residue of n, where n is composite. All the factors of the above expression are Legendre symbols. A Legendre symbol has value 1 if q is a quadratic residue modulo p_i, otherwise it is -1 if q is a quadratic nonresidue modulo p_i. Now, when the Jacobi symbol is 1, it could be that all of the Legendre symbol factors are 1(which implies the number is a quadratic residue), that is they are quadratic residues, but it could also be that there are an even number of Legendre symbols with value -1(which implies it is a quadratic nonresidue). This last point is critical: when someone does not know the factors of n, he can not[2] know whether it is a quadratic residue or not.

3.8.3 Quadratic Residuosity Assumption

Informally, the quadratic residuosity assumption states that it is hard to distinguish quadratic residues modulo a composite number n from quadratic non-residutes modulo n with Jacobi symbol 1. The formal definition can be found in [GM84].

If n is an odd prime, on the other hand, it is easy to decide whether something is a quadratic residue or not: If $q^{(n-1)/2} \bmod n = 1$ then q is a quadratic residue. If $q^{(n-1)/2} \bmod n = -1$ then q is a quadratic nonresidue.

The first of the two above rules is known as Euler's criterion.

3.8.4 Blum integer

An integer x is a Blum integer iff $x = p^{k_1}q^{k_2}$, where p and q are different primes, such that k_1, and k_2 are odd integers, such that $p = q = 3 \bmod 4$.

3.8.5 Random Oracle Model

In the Random Oracle Mode(ROM) one replaces all hash functions by calls to the "random oracle".

Citing [BR93]:

Definition 3.9 *A random oracle R is a map from $\{0,1\}^*$ to $\{0,1\}^\infty$ chosen by selecting each bit of R(x) uniformly and independently, for every x. Of course no actual protocol uses an infinitely long output, this just saves us from having to say how long "sufficiently long" is. We denote by 2^∞ the set of all random oracles. The letter R will denote the generic random oracle, while $G : \{0,1\}^*$ to $\{0,1\}^\infty$ will denote a random generator and $H : \{0,1\}^k \longrightarrow \{0,1\}^k$ a random hash function. Whenever there are multiple oracles mentioned, all of these are independently selected. Via all sorts of natural encodings, a single random oracle R can be used to provide as many independent random oracles as*

[2]atleast currently nobody has shown an efficient method of how to do this

one wants. As usual the oracles provided to an algorithm are indicated by superscripts. Sometimes the oracle is understood and omitted from the notation.

The ROM is used when the security of some system can not be proved in the standard model(i.e. a proof without the use of a random oracle). In a real system the random oracle is replaced again by a a good hash function, since in reality there are no random oracles. One final note: it is possible to construct a scheme which is secure in the random oracle model, but insecure outside it. See [CGH04] for further details about this result. We do not discuss this or the ROM further.

Chapter 4

Biometrics for border control

Written by *Mark Suurmond and Roderick Jansen.*

Currently people's identity is verified at the various national borders by comparing their face to a picture in their passport. To enforce border control it would be nice to be able to use biometrics as an extra check. Biometric templates however should not be stored in publicly readable places (like a piece of paper). An explanation why biometric templates should not be stored in public places will be given later. From this point on there are two obvious ways to link a biometric template to a passport.

1. Put a smart card in each passport. This smart card will assure the template can be stored and used for identification, but not give away the template.
   Because this option would require each passport in the world to have an embedded smart card this option will be way to costly to be realistic.

2. Put all the biometric templates in a huge database and give each border crossing access to it.
   This is not a very nice option either since there will have to be transmitted a lot of confidential data (the biometric templates) across the world.

Because the obvious methods don't seem very preferable an other method has been created.

4.0.6 Properties of the proposed solution

The proposed solution is a solution in which:

- Biometrics can be used to check weather or not a passport belongs to the person presenting it

- No confidential data has to be sent around the globe.

- No smart cards are needed in passports

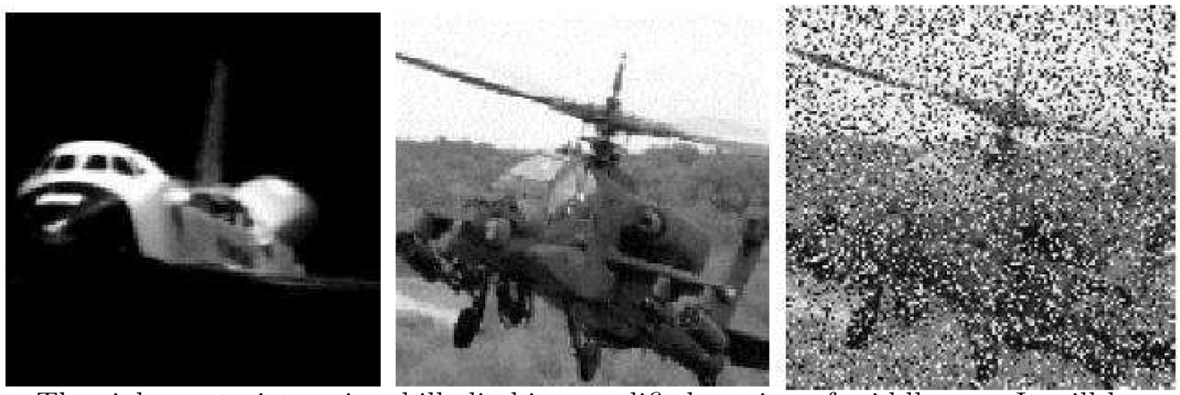

The rightmost picture is a hill climbing modified version of middle one. It will be recognized as the leftmost picture despite the fact that you can see that it is an Apache helicopter and not a Space shuttle. Those pictures were taken from `http://www.bioscrypt.com/assets/security_soutar.pdf` [Sou02]

Figure 4.1: Graphical example of a hill climbing attack

- Optionally a public key infrastructure can be used to verify that the passport indeed belongs to the person presenting it.
  Using this public key infrastructure will generate data traffic between all border crossings and the PKI. The difference with sending biometric templates is that in this case the data is not confidential.

4.0.7 Why biometric templates should not be stored in publicly readable places

Biometric templates are vulnerable to a masquerade attack. If you have access to the biometric system, and you have someone else's biometric template you can do the following:

1. Check how much an image of yourself differs from the template.
2. Change a few random pixels in the image of yourself.
3. Check again with the modified image.
4. If the modified image is worse then the original return to step 1.
5. If the modified image is better see if it exceeds the threshold value of the biometric system (Does it return a positive identification?) you are done.
6. If the modified image is better then the original but still below the threshold value repeat from step 1. Use the modified image.

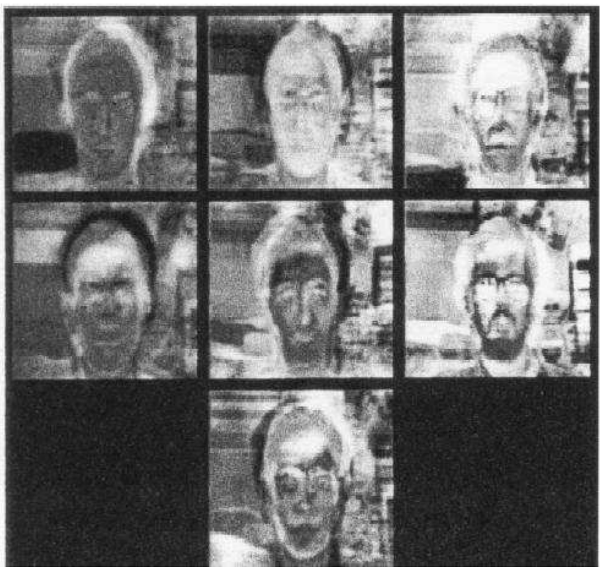


Figure 4.2: Some graphical examples of eigenfaces

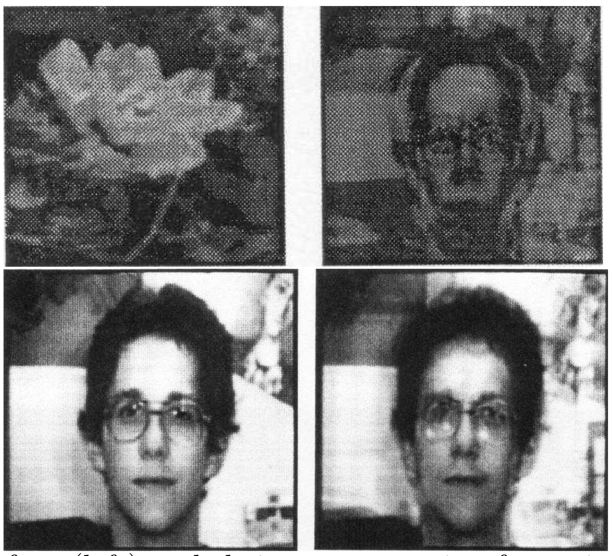
After a while you will get an image that closely resembles your own image but in fact matches the template you want to attack. By using this image as input for the biometric system you can identify yourself as the person whose template you have attacked. Because this attack keeps on improving the picture step by step it is called a hill climbing attack. For a graphical example of this hill climbing attack see figure 4.1. The Hill climbing attack and a method to reduce its feasibility are explained in more detail in [Sou02].

4.1 The biometrics

One of the things you need to be able to do all of this is biometrics. In the scheme two types of biometrics are used. Those are face recognition and fingerprints. For both there exist techniques based on feature representation. (e.g. How far are the eyes apart? Are both ears equally big?)

For face recognition there has also been developed a new method based on linear algebra. Once developed this method also appeared to be suitable for fingerprint recognizing. I will explain this method in a bit more detail. As said it is applicable for multiple image similarity measuring problems but since it was originally developed for face recognition I will discuss it in a face recognition context.

Face recognition is done on images of faces. To build a recognition system you will first need a set of training images. From those images a set of eigenvectors is created. For more information on how this is done see [TP91]. Those eigenvectors are called "eigenfaces" from now on. To get an idea what those eigenfaces are like see figure 4.2



A flower and a face (left) and their reconstruction from eigenfaces (right).
As you can see the image of a flower doesn't even remotely resemble a flower anymore when reconstructed from eigenfaces while the face has hardly changed by the reconstruction.

Figure 4.3: Projection of a flower and a face onto face space

it gives a graphical representation of some eigenfaces. Not all eigenfaces that can be generated from the training images will be used. As a consequence not all images can be generated by taking a linear combination of the eigenfaces.

The set of all images that can be generated by taking a linear combination of those eigenfaces is called "face space" At first it may sound disappointing that not all images are in face space, but if you look a bit closer it actually gives us exactly what we want. It is both possible to find out what the best way to represent a picture as a linear combination of eigenfaces is and to find out how close this constructed image resembles the original image. If the resemblance is small the original picture probably does not show a face.

If a reconstructed image is much like the original, the chances are high that it in fact is a face. To see this concept illustrated see figure 4.3. Two pictures of the same face will lead to a similar linear combination of eigenfaces.

To make all of the above perform better, or even just work, a number of preprocessing steps can or should be taken. I will not discuss those steps in detail, but to give you an idea what the preprocessing is about I will list a number of possible preprocessing steps.

- Illumination normalization (Make sure all images have the same lighting conditions, This has a huge impact on performance of the face recognition system)
- Normalizing the scale (Make sure each face is equally big)
- Translating (Place the center of the eyes at standard locations)

- Background elimination (Make sure the background of all images is the same in order to not let the background have any effect on the recognition process)

4.2 Phase 1 of the proposed solution

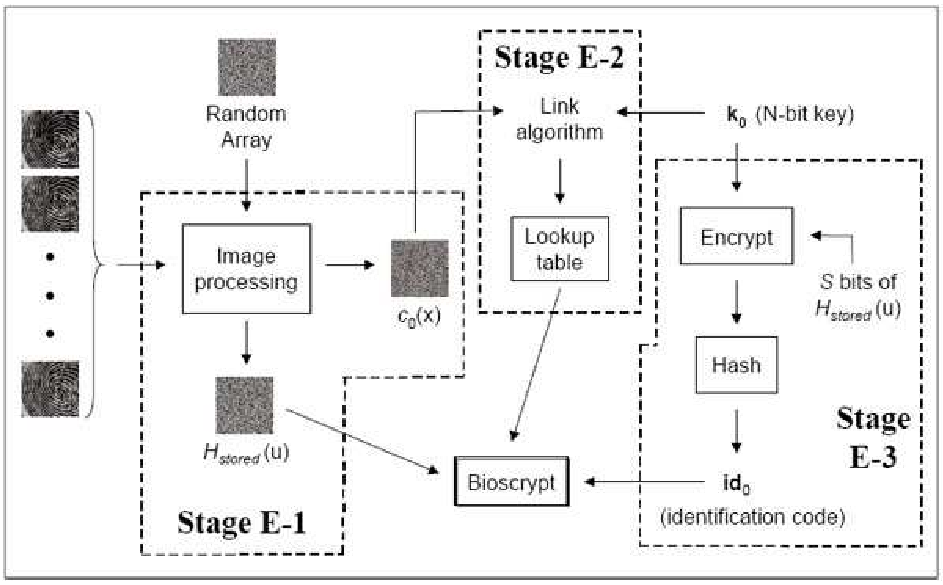
The proposed solution consists of two phases for identifying some one. Those phases are worthless without each other but to be able to explain the solution in an understandable way we explain them one by one. Phase one is actually a modern and complicated way of doing what we already do for many years. What we are going to do is: "Put a picture in a passport so an inspecting officer can check if the passport belongs to the person holding it." Because we need to cryptographically sign and link things together later on, just putting a picture in the passport will not do here. The picture needs to be digital and fit on a 2d barcode together with many other things. To this end we use the face recognition method described in section 4.1. in a non-standard way.

Each border crossing as well as the various passport issuing departments have access to a biometric system that uses the same set of eigenfaces called P_{t_0}. An eigenvector representation called B_{t_0} of the passport holder's face is printed on the passport in barcode. The inspecting officer at the border can then reconstruct the face of the passport holder by combining (P_{t_0}) and (B_{t_0}). This combination can be showed to the officer so he can compare it to the passport holder's face. The actual recognizing part of the face recognition system is not used here. The comparison of the reconstructed face with the real one is left to a human inspector. Using the biometric template here would be vulnerable to a hill climbing attack because the template (B_{t_0}) is stored in a publically readable way.

4.3 Phase 2 of the proposed solution

This chapter is about further securing biometric data in a passport as described in [KM05]

As the preceding chapters described, putting a biometric template in a passport is not enough as described in the hill-climbing attack. We thus want an additional procedure of verification, one which can't easily be altered by an attacker. We still use the facial template so a border control officer can identify the person by looking at the reconstructed image of the facial-template. But we also add the use of a fingerprint as a second level of verification. The fingerprint is stored in such a way that a partial private key can be derived from it when it is compared with the real fingerprint.



The three steps to create a secure block of data. This scheme is taken from [SRs+99]

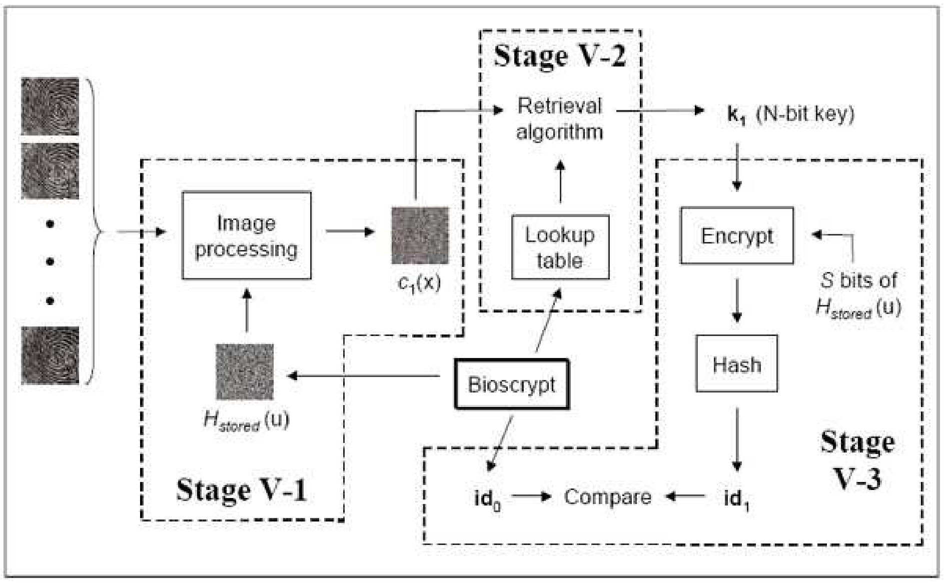
Figure 4.4: The procedure to create a BioScrypt

4.4 How to store a private-public key with use of biometric data

A key for decrypting data can be secured by a password. The problem here is that storing the key in a safe way is (at least) difficult, especially when the key is stored on the users local computer (or on a password). What can be done is hiding the cryptographic key in the biometric template using a bit-replacement algorithm. When the user is successfully authenticated the key is simply extracted from the appropriate bit locations. The obvious problem here is that when an attacker knows the bits which contain the key, he could extract it and place it in his own biometric data.

A new technique for securing a key using biometric has been developed by Mytec Technologies Inc. The key is linked with the biometric template and is later retrieved with biometrics during verification. The key is also completely independent of the biometric data, so when the key is found by an attacker it can easily be updated and the user will still be able to use his/her biometrics. This also gives the advantage that a key can be adapted as an extra safeguard.

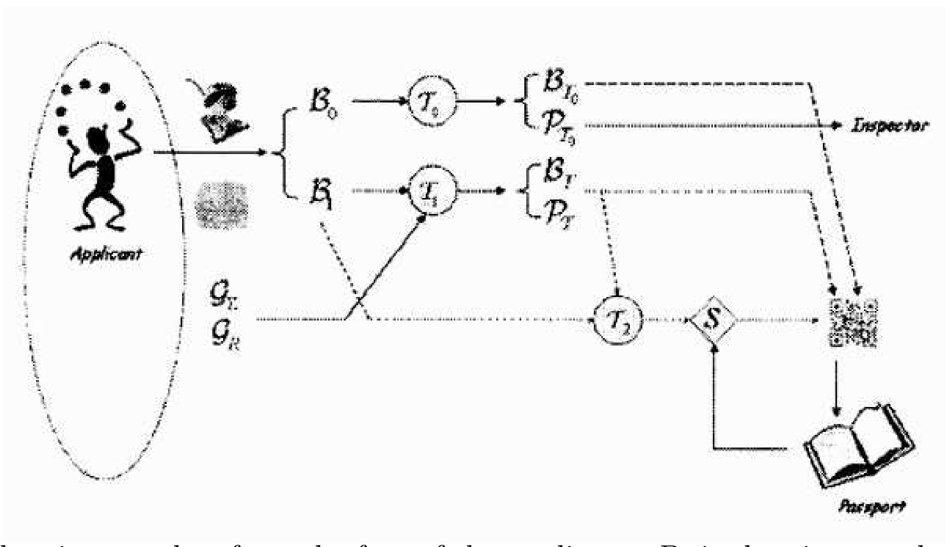
During the creation of a passport, the biometric template is combined with a digital key to create a secure block of data, which Mytec Technologies names a BioScrypt. The digital key can then be used as a cryptographic key further in the process. Through the BioScrypt the digital key and the fingerprint can't individually be extracted. When a user sends its fingerprint to the BioScrypt, it get his/her key back. So the BioScrypt is a secure method of managing keys.



This scheme shows the procedure which is followed when a user gives his fingerprint and he/she is verified. And is taken from [SRs+99]

Figure 4.5: How to use the BioScrypt to verify a cryptographic key

- **E-1** Image processing:
  A series of input images of fingerprints is combined with a random array to create $H_{stored}(u)$ and $C_0(x)$.
- **E-2** Key linking:
  Link a cryptographic key K_0 to the pattern, $C_0(x)$, through the link algorithm.
- **E-3** Identification code creation:
  Create an identification code, id_0 which is derived from the key, K_0
- **V-1** Image processing:
  Combine $H_{stored}(u)$ from the BioScrypt with a new series of input fingerprint images to create an output pattern $C_1(x)$.
- **V-2** Key retrieval:
  Extract a key K_1 from $C_1(x)$, using the retrieval algorithm.
- **V-3** Key validation:
  Validate K_1 by creating a new identification code id_1 and compare it with id_0.



B_0 is the picture taken from the face of the applicant. B_1 is the picture taken from his/her fingerprint. This picture is taken from [KM05]

Figure 4.6: THE PROCEDURE TO GENERATE A BARCODE FOR A PASSPORT

4.5 Generating a Signature and a public-private key pair

When a person wants to get a passport, the scheme in figure 4.6 is used. In the first chapter the first phase was already described. We concentrate here on the second phase.

The Biometric information of a fingerprint is transformed together with a G_e (which is a digital signature scheme e) and with G_r. (which is a random number generator) We then get a B_t and a P_t out of this transformation. B_t is the biometric template of the fingerprint and is printed in the barcode. P_t is a pair of $\{e_2$,M$\}$ where e_2 is a part of the public key.

So we have:

- $T_1 : [G_e(1^\alpha), G_r(1^\beta), B_1] \rightarrow [B_t, P_t]$
- $T_2 : [B_1, B_t, P_t] \rightarrow [G_e]$

A public key is generated at the creation of the passport by using a RSA scheme. $G_e(1^\alpha)$ outputs a public-private key pair $\langle e, N\rangle$ $\langle d, N\rangle$. e_2 is then random chosen in such a way that:

$$e_2 = e \cdot e_1^{-1} \bmod o(N) + \beta o(N).$$

Then e is discarded. e_2 becomes the public part of the public key and is around 1024 bits $(log\beta + \alpha)$. e_1 is about 128 bits and is woven into the fingerprint scheme in such a way as described in the preceding chapter. The use of a random number generator has the advantage that a compromised passport can easily be modified by a passport-signer. The user's possession $P(= (B_{t0}, B_T, P_T))$ is defined as $B_t = \{H_{stored}(u), L\}$ and $P_t = \{e_2, N\}$ while B_{t0} was obtained in phase 1. L is the lookup table, so we know were too look for the key e_1.

Transformation 2 gets input form B_1 (the biometric information of the fingerprint) B_t and P_t and generates G_e (which gives the public key (e)). Remember that P_t included e_2 and e_1 is the 'difference' between B_t and B_1. Now $\langle d, N\rangle$ (G_e) is used to create a Signature **S**.

The passport creator raises information M (say, the identification contents in a passport) to the power d for obtaining the corresponding signature $\sigma = M^d$ mod N then d is discarded. The signature is also used to sign the facial feature representation (B_0), the fingerprint representation with the key e_1 in it (B_t) and the public part of the public key($P_t\{e_2$,M$\}$). So that if any data in the barcode is tampered with, it becomes obvious to the controlling officer. The signature, B_t, P_t and B_{t0} are printed on the passport in a 2D barcode.

4.6 Verification at the border

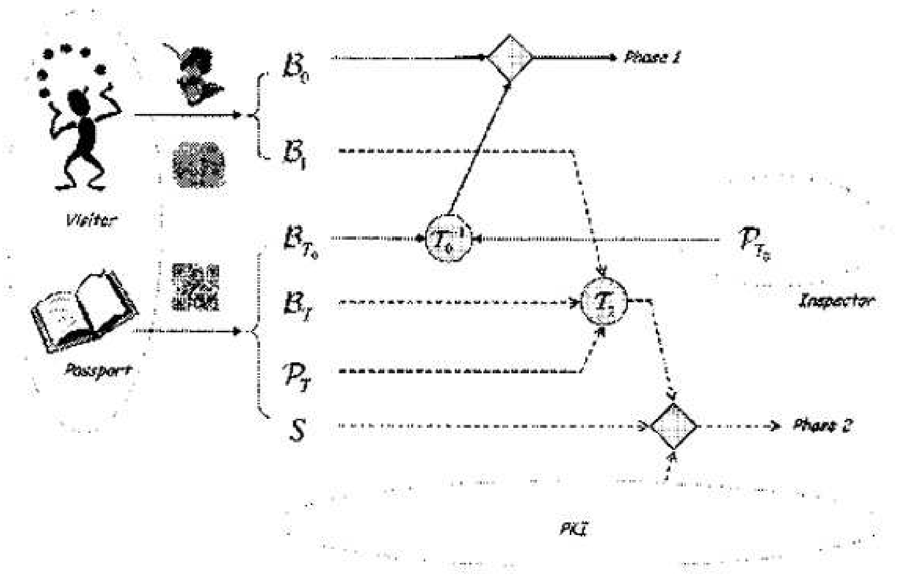
After the facial template is confirmed by a Inspector, the user is asked for his/her fingerprint. (B_1) As described in figure 4.7.

The result is then combined with information for the 2D barcode. [B_{t0}, B_t and P_t] gives $[H_{stored}(u), L, e_2, N]$ and from the difference between the given fingerprint and the stored fingerprint a partial key (e_1) is looked up.

Now the Signature **S** is raised to the power of e_1 and e_2 to verify $M = ((S)^{e_1})^{e_2}$ mod N. The PKI can also be asked if e_2 is a valid key, but this is optional. If a border control officer has enough confidence in the anti-paper-tempering qualities of the passport itself, he can verify the keys. If he wants to be more certain he can consult a PKI whether the generated key is known as a valid one by the PKI. If this results in a positive outcome the use can cross the border (or not).

4.7 Conclusion

The proposed schema does succeed in keeping the biometric templates secret without needing smart cards. A small note that has to be made is that the schema provides little protection against malicious border control machines. If an attacker builds a machine that looks like a border control identification machine he could capture the passport barcode data and the biometric template. This might enable him to generate fake passports. Whether or not this will be possible depends strongly on the implementation.



The solid line is Phase 1 and after verification by the border control officer, Phase 2 is used to control his/her fingerprint and Signature **S**. This picture is taken from [KM05].

Figure 4.7: THE PROCEDURE TO CONTROL A USER WITH HIS/HER PASSPORT

The schema uses quite some things in a non standard way like for example a private share of a public key and a face recognition system that relies on a human being recognizing the reconstructed picture. This makes the schema rather complex and a bit confusing. Because this schema is not really simple mistakes are easily made so implementation will have to be done with care.

Because the chain is as strong as the weakest link in is of extreme importance that the passport issuing agency is trustworthy and accurate. There is of course no technical solution for a negligent or even vicious passport issuing agency

Chapter 5

Human Interactive Proofs

Written by *Clement Gutel, Haiko Schol.*

5.1 Introduction

In 1997 Alta Vista was looking for ways to block or discourage the automatic submission of URLs to their search engine. This free "add-URL" service is important to AltaVista since it broadens its search coverage. Yet some users were abusing the service by automating the submission of large numbers of URLS, in an effort to skew AltaVista's importance ranking algorithms. More generally nowadays, most of the online services are suffering from abuse by programs made by malicious programmers. The problem is that such program can impersonate humans to register for free mail accounts, send spams, vote on online polls, post message in blogs, make advertising in chat rooms, abuse services and do a lot of additional various attacks.

Efforts to defend against such attacks have, over the past several years, stimulated investigations into a new family of protocols : Human Interactive Proofs (HIP). HIPs allow a person to authenticate herself as human. HIPs are also known as CAPTCHAs. CAPTCHA stands for "Completely Automated Public Turing test to tell Computers and Humans Apart".

Classical cryptography has often factored humans out of the equation: when we say Alice and Bob can communicate securely (or authenticate, or sign, or perform a zero knowledge proof), we really mean Alice and Bob's computers can communicate securely. This has resulted in humans being a major security hole in practice.

Current authentication methods for computer systems, web pages, or monetary transactions are weak in some ways. Passwords, social security numbers, phone numbers, and PINs all can be easily stolen or shared and are routinely known to others, e.g. system administrators, who may abuse them. Biometrics require special hardware and do not provide universal coverage. More strikingly, biometrics require trusted terminals.

The idea of HIP is to create a program that can generate a very big number of different challenges that are easy to solve for humans and hard for computers. This must be done cheaply and on-the-fly. When a user wants to use a service protected by a HIP, he/she is asked to solve the challenge before accessing the service.

5.2 Reading-based CAPTCHAs

5.2.1 The challenge

The challenge asked to be solved by the user is quite simple in the case of a reading based HIP. An image containing a sequence of characters is generated, this image is then deteriorated through various cluttering and deformations. To pass the challenge, the user simply has to type in the sequence of characters.

5.2.2 Reasons for using reading-based CAPTCHAs

At this date, reading-based HIPs are the most obvious favourites and suitable HIPs. Here are several reasons for their popularity:

- Optical character recognition (OCR) is a well studied field and the state of art is known
- Characters were designed by and for humans, and humans have been trained at the task since childhood
- 8 characters on the keyboard span a space over 1000 billion solutions
- Localization issues are minimal using western characters and numbers, i.e. almost every human that can read, knows how to read western characters.
- The task is easy to understand without much instruction.
- Character-based HIPs can be generated quickly.

5.2.3 Implementation

There are a lot of open source implementations of basic HIPs generators for the usual programming languages. However, some people argue that the generation of HIPs should be done with a open source webservice because it has several advantages:

- It becomes possible for anyone to protect his/her site/online service without bothering with the details of the implementation: it is an application independent task.
- HIP generators are made available on every platform.
- Updates on the generators can be deployed widely very easily.

One of the several web service offering HIP generation is http://captchaservice.org For a basic use, the client simply sends an http request to the webservice, which replies an XML response containing a link to the generated HIP image. In more advanced use, the client can choose the language model and various parameters that influence the level of distortions applied on the text.

Several design choices have to be made when implementing an HIPs, like the character set, the applied transformations, the cluttering strategy (arcs, lines), the wrapping strategy and the chosen textures. Moreover, a language model must be chosen, this could be english words, random characters or even english-like words created by phonetic generators. Most online HIPs do not use more than 8 characters.

Of course, all these design choices influence the human friendliness of solving the HIP, and the difficulty of breaking the HIP.

5.2.4 Security

Cracking a reading-based HIP

The algorithm used to create the HIP is usually made public, though it may be covered by a patent. This is done to demonstrate that breaking it requires the solution of a hard problem in the field of artificial intelligence (AI) rather than just the discovery of the (secret) algorithm, which could be obtained through reverse engineering or other means. Moreover open source programs can benefit from collaborative help to improve the program and to find bugs. Simple security measures can be taken like tracking large numbers of incorrect guesses coming from the same IP adress.

However, computer vision has made a lot of progress and the easiest HIPs can already be broken.

Reading-based HIPs cracking algorithm typically have two challenges: the character segmentation and then the character recognition.

- Segmentation:

The segmentation challenge is about identifying character locations in the image in the right order. This is made difficult by the random location of characters, background textures, foreground and background lines, and arcs.

- Character recognition:

Once the segmentation problem is solved, solving the HIP is a pure recognition problem that can efficiently be solved using machine learning. Most of recognition engines are based on neural networks. Once the engine is sufficiently trained, it succeeds efficently. Given an image of a character under a variety of distortions (translation, rotations, elastic deformations, scaling) and clutters (arcs, lines, textures, back/foreground), the goal is to recognize the character. Experiments done in [1] reveal that with a training set of 90,000 words, and a validation set of 10,000 words, the engine is able to solve the problem efficiently (around 85% success). Therefore, the real breaking challenge is solving the segmentation problem.

Attemps to break HIPs is not something new. The formely used EZ-Gimpy HIP generator was broken by Mori and Mali with 92% success.

Other attacks

Spammers have found a way to circumvent this restriction: simply present the captcha to a human user under false pretenses, and use the human's response to obtain the e-mail account.

To do this, the spammer must control a Web site to which human users wish to gain access, for instance, a pornography site. When a user goes to the spammer's porn site, the server starts a new account registration at the free e-mail provider. It downloads the provider's captcha and presents it to the user as a captcha for access to the porn site. The user, not knowing that the captcha is recycled, provides the correct response, and the spammer's software can then complete the e-mail account registration.

It may also be possible to subvert captchas by relaying them to a sweatshop of human operators who are employed to decode captchas. Nonetheless, some have suggested that this would still not be economically viable.

5.2.5 Human friendliness

Human friendliness encompasses both a) the visual appeal and annoyance factor of a HIP and b) how well it utilizes the difference in ability between humans and machines at solving segmentation and recognition problems. Many of the design choices that make HIPs human friendly tend to reduce security and vice-versa. However this is not always the case, some attributes like colors, textures and size have negligible effect on security while they can significantly improve appeal. A study conducted on seventy six users in [1] has shown some interesting results than can help to build better HIPs, i.e. harder for computers to solve and easy for humans. The experiment is quite simple, some HIPs were generated using different design parameter choices, and users were asked to solve them. According to the level of accuracy of users, reasearchers have come to the conclusion that while some kind of image deformations make the task a lot harder for humans, some other almost do not affect the difficulty (again this is only for human). This has lead to the knowledge of the optimal parameters settings to produce HIPs as hard as possible for computers and easy for humans. Interestingly, the study has shown that computers are better at character recognition, but humans are far better at segmentation. Indeed, when performing some elastic deformations on single character (local wrapping), or on the whole image (global wrapping), humans tend to have difficulties recognizing the characters, while computers perform well. On the other hand, adding some lines and arcs in the pictures all around characters does not affect the human difficulty to segment the image whereas it confuses computers very much. They take those lines and arcs for characters during the segmentation process. Therefore, better HIPs can therefore be created, increasing the segmentation difficulty.

5.3 Other approaches for building CAPTCHAs

As mentioned above, the most common form of human interactive proofs, visual text-based CAPTCHAs suffer from inherent shortcomings. Therefore, in addition to research on how to improve those HIPs, efforts are made to develop other forms of HIPs. Because visual text-based CAPTCHAs are around for a much longer time than alternative methods, the latter are not yet as advanced as the former. In this section, some alternative approaches to develop effective human interactive proofs will be presented.

5.3.1 Image-based CAPTCHAs

An approach that is quite similar to visual text-based CAPTCHAs are visual HIPs that require the user to recognize objects displayed in an image. In such a test a picture depicting a certain object is presented together with names of different objects and the user has to choose the right name. Alternatively, some pictures are presented and the user is asked to choose the one that does not fit in with the others. This is a sort of logic puzzle, that might improve resistance to attack. If an attacker is able to map a given image to the correct class, he might fail when the class is only given implicitely by the group of images presented. Again, this kind of HIP can not be solved by users who have impaired vision abilities. Also very problematic is that the solution space for these CAPTCHAs is very much smaller than that of CAPTCHAs displaying a string of characters. By requiring the user to choose between a few different possible solutions the probability to pick the right one by guessing is significantly higher than it is with text-based HIPs. Even if a text-based human interactive proof only displays one lowercase letter, the probability to guess the correct one would be 1/26, whereas the probability to pick the right object out of 5 possible choices would be 1/5. This weakness alone might make brute-force attacks on CAPTCHAs based on object recognition feasible in certain scenarios. One of those would be automated creation of email accounts with free webmail services.

Another issue, which compounds the problem of a small solution space, is that images of existing objects are not easily generated by an algorithm. It is neccessary to compile a library of images in advance from which the challenge is chosen. If this library is too small, it might even be feasible to solve all, or most of the CAPTCHAs beforehand and simply match the given challenge to a database of correct solutions. Additionally, the more images of objects are in the library and the more different choices are presented for a given challenge, the less human-friendly such a CAPTCHA becomes. Again, the possibility of introducing a cultural bias has to be considered. It must be assured that all the images in the database depict objects that are equally well recognized by all cultures and demographics. Furthermore, there are cognitive disabilities that prevent the patient from recognizing objects by looking at 2-dimensional representation. This adds to the obvious accessibility constraints that can be identified in visual CAPTCHAs.

A special case of CAPTCHAs based on object recognition is to use images of human faces. This approach addresses some of the above mentioned shortcomings. Face recognition is not specific to any culture. Therefore it is equally easy to solve for people from any part of the world. Furthermore, pictures of human faces can, to an extend, be generated automatically. Law enforcement organisations use methods to generate images of human faces from descriptions of crime victims, for example. It is still neccessary to have a library of parts of a human face that are assembled by an algorithm, but these are readily available for applications such as generation of photofit pictures. Additionally, a much larger space of possible challenges is available with this approach compared to a fixed set of images depicting different kinds of objects, because the parts of a human face can be combined in various ways and also slightly modified by image manipulation algorithms. This may be true for pictures of objects as well, but it could be argued that the challenge "which of the images shows a human face?" is more human-friendly than "what kind of object is displayed?", given slight distortion of the presented images.

In such a test, many pictures would be presented as a challenge, with only one of them depicting a human face with the right amount of facial features (i.e. one nose, two eyes, etc.) and the other pictures depicting a random combination of the parts of a human face from the database. Unfortunately, this kind of HIP does not solve all of the problems encountered with CAPTCHAs based on object recognition. The first problem that comes to mind is again the issue of accessibility for people with impaired vision. Furthermore, the resistance to brute-force attacks is only marginally better compared to object recognition challenges. The user still has to pick one image from a small set of possible solutions, so randomly choosing pictures would not be as ineffective as generating random input for text-based CAPTCHAs.

Yet another image-based CAPTCHA can be build by using handwritten text. The problem of algorithmically recognizing handwriting is much harder than the problem of recognizing text written in a computer font. Modern handwriting recognition software that is used in devices like PDAs or tablet PCs depends upon being active during the user enters text. It is taken into account how fast, in what direction, etc. the user applies the stylus to the touchscreen. Results are not as good if the software only has an image of the written word. This means, that handwriting-recognition CAPTCHAs are probably harder to break than common text-based CAPTCHAs. Automatic generation of new challenges is possible by combining single letters from handwritten words. The image can also be manipulated in the same way conventional CAPTCHAs are in order to further increase variance. If not done careful, this can make the text very hard to read for humans as well. Results from research in Gestalt psychology can be applied to mitigate this problem.

5.3.2 Audio text-based CAPTCHAs

An obvious alternative to tests that require the user to recognize text in form of an image are those, that require the user to recognize text in form of an audio signal. Speech recognition is still considered a hard problem of Artificial Intelligence, although significant advances have been made in recent years. The signal might be mixed with noise or otherwise distorted to improve resistance to attack, but there is a limit to how far this can be taken without generating too many false negatives. Although in general CAPTCHAs relying on audio signals meet the requirements for any sort of human interactive proof in that they are easy for humans and hard for computers, they exhibit similar shortcomings as visual text-based CAPTCHAs. First and foremost, they obviously do not meet accessibility criteria. Just like a blind person can not decode an image that contains text, a deaf person can not decode an audio CAPTCHA. To alleviate this problem, it is neccessary to deploy CAPTCHAs that are offered in both forms, catering to visual and auditorial perception. But this does not solve the problem completely, since there are also people who can neither see nor hear. This means, that audio text-based CAPTCHAs do not meet accessibility requirements as put forth by the World Wide Web Consortium, even if they are combined with HIPs that utilize other modes of perception. This is a very serious problem that is possibly shared by all CAPTCHAs, since they all require a specific mode of perception or cognitive abilities, such as logic reasoning.

Another, equally problematic shortcoming of audio text-based CAPTCHAs is that they might introduce a cultural bias. It is significantly easier for a user to solve such a CAPTCHA if the speech synthesis used to produce the audio signal uses the same pronunciation rules as the native language of the user. If this problem is to be avoided, it is neccessary to develop synthesis rules for all languages that are to be used in the test. In other words, the amount of work that must be done increases linearly with the number of languages that need to be supported. For visual text-based CAPTCHAs that simply generate random character strings, the amount of work does not increase with the addition of new languages to a website.

5.3.3 Logic-based CAPTCHAs

A slightly different way to build a HIP is that of using questions that must be understood and correctly answered by a user. Instead of a problem of perception such a CAPTCHA constitutes a problem of understanding natural language and reasoning about meaning in order to infer a logic conclusion. Therefore, such a test includes two problems of Artificial Intelligence that are considered at least as hard or harder than the problem of computer vision or speech recognition. The first problem that needs to be solved requires natural language processing , that is parsing the challenge and extracting the “meaning” of the question. The word “meaning” has many highly philosophical implications and it could be argued, that a computer can never understand the “meaning” of a proposition in the same way that (most) humans do. Concerning the problem of answering a logic riddle type of question these philosophical musings need not neccessarily be considered. There are quite sophisticated logic programming tools, such as PROLOG , that allow a computer to infer answers from a database of facts and rules. A very simple example would be to enter the following rules into such a system:

- A brother is a sibling.
- A sister is a sibling.

and pose the following logic question: “Alice has 2 brothers and 3 sisters. How many siblings does she have?”. By applying the given rules, the logic system can easily infer the correct answer without knowing the “meaning” of the word sibling. What is not so easy is the natural language processing part. That is, given the question formulated in plain english, the computer has to find out what the correct query is that needs to be computed by the logic system. Both problems are topics in the field of Artificial Intelligence that are actively being researched. Currently it can be assumed, though, that the techniques developed in the course of this research are not as effective for breaking logic-based HIPs as computer vision techniques are for breaking visual text-based CAPTCHAs. But specific examples of visual text-based CAPTCHAs can be found that have been improved in terms of security by taking known attacks into account. Therefore it might not be as clear and simple to assess which of those methods for implementing human interactive proofs is more effective in practice.

Apart from potential security benefits, logic-based CAPTCHAs are not bound to a specific mode of perception. They are given as plain text, in the same way as the content of a website. Therefore, accessibility is greatly improved compared to conventional HIPs, because every user that can make use of the content of the website can also access the challenge he needs to solve. Accessibility is still not optimal. There are cognitive disabilities that prevent the patient from understanding natural language as well as those that prevent the patient from producing meaningful propositions. On a website that does not require these abilities in order to be useful such a CAPTCHA would impose an accessibility barrier. Depending on what kind of logic riddles are used to create the challenge it might be the case that the CAPTCHA is basically a very limited form of IQ test. This could also be seen as an accessibility barrier, without considering specific cognitive disabilities. Finally, logic-based CAPTCHAs can not be generated automatically. This problem has also been seen with many other approaches already. It is neccessary to create a database in advance and also to constantly add new content to prevent replay attacks.

5.3.4 Collaborative filtering CAPTCHAs

Collaborative filtering is a technique originally developed for creating a recommender system that scales well with the number of items that are to be included. With the phenomenon of information explosion conventional recommender systems that require a human to give recommendations are not very useful anymore. In a small book or music store the owner might know a large amount of the content and can estimate the taste of his customers based on former purchases. For vast amounts of content as for example offered by amazon.com, this approach does not scale. Therefore, algorithms have to be implemented that exploit an existing knowledge base about the taste of users in order to infer useful recommendations for future purchases.

There are different ways to realise this. One possibility is to have the users explicitly rate items. A naive approach would simply average over those ratings and, given a small number of ratings from a user for whom a prediction is to be calculated, look for items that match the given ratings. This approach fails to work under the premise of diversity of taste. What is done instead is to only consider those users from the knowledge base whose rating patterns match that of the user who is asking for a recommendation. Recommendations are then computed from this strict subset.

During the second international workshop on human interactive proofs, a method for using collaborative filtering as the basis of a HIP was presented [CT05]. As opposed to the other CAPTCHAs mentioned in this paper, this CAPTCHA presents the user with a challenge that can not be solved by a correct answer. Instead, the collaborative filtering CAPTCHA determines whether the answers given by a user are consistent with the answers of previous users. From the side of an attacker, the problem to solve is to automatically predict human behaviour (such as ratings based on taste) without having access to the database of patterns that are known to reflect human behaviour. The requirements that all CAPTCHAs have in common also hold for collaborative filtering, of course. Two of them can be expressed for the special case of collaborative filtering in the following way:

- The prediction of a users rating must be accurate. If it is not, human users will be rejected. This would violate the requirement of human-friendliness.
- The data in the knowledge base must be evenly distributed over the range of possible ratings. If this is not the case, an attack might be successful that always chooses the rating that occurs most often.

Rating items of art, such as books, movies or songs based on personal taste is only one of many possible data source. Other examples given in [CT05] include what kind of emotion a piece of art evokes in a human. Another proposed metric is to have users rate how funny they think a joke is. Different kinds of jokes, for example puns and offensive jokes, cater to different kinds of humor. Using these data sources to build a knowledge base is quite a different task than developing an algorithm to generate images containing distorted text. Such an algorithm needs to be developed only once, maybe with subsequent modifications to counteract attacks. Theoretically it only takes one person to write such an algorithm. Creating a database on which collaborative filtering CAPTCHAs can be carried out requires many humans, rating a sufficiently big dataset before the system can be used. This may require a lot of time, for example if the items to be rated by the users are movies. Building the initial database in this way constitutes the first phase of a collaborative filtering CAPTCHA.

The second phase is made up of the actual usage of the system, where users are given a small subset of the items in the database to rate. The system then calculates predictions for other items that are then presented to the user. If the ratings given by the user are close enough to the predictions, the user is considered to be human. After admitting the user to use the service, the database needs to be extended, to prevent replay attacks on a static dataset. This is done by giving the user new items to rate. After a sufficient amount of ratings is stored for the new items, they enter the database and are used for the testing phase. This last phase of extending the database is called reseeding.

For predicting ratings by means of collaborative filtering an often applied method called Singular Value Decomposition (SVD) can be used. The idea is, that items that are to be rated have certain features that influence the rating a user will give to an item. However, the user is not required to give ratings for every feature, but only a one-dimensional value per item. The outcome of doing this for many users and many documents is a rating matrix, in which a row corresponds to one user and his ratings for a number of documents. Additional input to SVD is made up of a matrix V describing the amount of each feature in each document and another matrix S that is only non-zero on the diagonal. The matrix S determines how important each feature is. The output of calculating the SVD denotes how much the user likes a certain feature. By combining this output with the information stored in V, predictions can be made how a user will rate a new item. The resulting predictions can be improved by applying nearest-neighbor algorithms before calculating the SVD. A row containing all known ratings of a user can be seen as a preference vector of that user. Searching the dataset by applying a nearest-neighbor algorithm a set of users is found whose preferences are close to the user who is to be tested. Two preference vectors are considered to be close if the cosine between them is close to 1.

In [CT05] two experiments are decribed. One uses an existing database of rated jokes called Jester. The second experiment uses art images and requires the user to enter what kind of emotion the images evoke. Some of the images were taken from an online art gallery and some were generated random art images. Results from the first experiment showed, that using the whole dataset to compute the SVD does not yield accurate predictions. The average error of predicted ratings was larger than the error of randomly generated predictions. By only using the 10 nearest neighbors, predictions were significantly improved. The collaborative filtering algorithm developed by the designers of the Jester system, called Eigentaste, achieves a much lower average error. This algorithm was not implemented in the collaborative filtering CAPTCHA, but the existing data from the Jester project suggests, that there is potential for further improvement.

The second experiment only included a very small number of items to rate and a very small number of human participants. But under these limited circumstances it already became clear that randomly generated images are not useful for this application. Users had difficulty in rating the random images, because they did not invoke much emotion at all. The two experiments hint at requirements of the dataset for a collaborative filtering CAPTCHA. Using jokes, as in the first experiment has the problem that jokes can not be generated automatically and constant effort is required in order to expand the dataset. On the other hand, randomly generated images are not sufficently evocative to be rated based on emotional reaction.

5.3.5 Alternatives to CAPTCHAs

This section contains a short overview of ideas for human interactive proofs that don't fit the definition of a CAPTCHA. In general terms, a CAPTCHA verifies that the user knows something. Another approach might verify that a user has something. One way to do this requires that a public key infrastructure is established where public keys are signed by a trusted certified authority. This could be achieved by having every citizen of a country generate a keypair with the public key signed by the government and stored in a central key server. This would be more or less equivalent to issuing id cards to citizens as is already common in many countries. The government (or another trusted party) would then guarantee, that the key owner is a human. Therefore a human interactive proof could be given by signing a challenge with the private key of the owner. This could also work as a proof of adulthood, if such a signed public key would only be handed out to adults instead of every citizen.

Such a system would require an enormous effort to implement on a country-wide scale and even more so on a global scale. Large scale PKI is not an easy task and the idea of creating something like this only for being able to carry out human interactive proofs is highly unrealistic. But if such an infrastructure were already in place, it could be used for this application. The most obvious drawback of this method is the lack of anonymity for the user. This problem alone is probably sufficient to rule out this HIP for most applications.

Another, even more problematic idea is to request a valid credit-card number from the user in order to verify that she is a human. The aforementioned problem of anonymity is compounded by the fact that the entity requesting the credit-card number must be trusted with highly sensitive data. Apart from that, it is obvious that not everybody has a credit-card. The two ideas mentioned so far are not very promising. A somewhat more interesting concept is that of not requiring a HIP at all at first. Only if a user exhibits a pattern of abusive behaviour towards the system, countermeasures are put into effect. One possibility to react is to limit the rate at which the user can utilize the offered service. For example, if a user sends many emails through a free webmail service and those emails get flagged by a spam filter the user could either be allowed to send only a limited amount of emails per hour or get locked out of the system entirely.

This way of dealing with abuse is certainly not applicable to all scenarios that HIPs are meant to be useful for. In the example of a free webmail service it is desirable to prevent a user from sending spam at all. In order to achieve this, the rate limiting could be put in place for all users. But this could interfere with legitimate uses of the service. The same holds true for the way to recognize patterns of abusive behaviour. It is a non-trivial problem to come up with a method to classify behaviour in a way that produces a minimal number of false positives.

The last idea to mention is that of using an old-fashioned turing test. One can imagine a text-chat session between a user and a live operator, where the operator asks questions that he makes up on the spot. This method can incorporate logic riddles or any other kind of meaningful conversation. An attacker would need to create an Artificial Intelligence that can not only parse natural language, solve logic problems from a predefined set and give answers in natural language. It would also need to be able to react to complete unpredictable input. It can be safely assumed, that such a program will not be created anytime soon. The most glaring problem of this approach is of course, that it is by far too expensive to have a sufficient number of live operators available 24 hours a day and 7 days a week. In an environment where the cost to do this is acceptable, it is most probably also acceptable for an attacker to spend the amount of money that is needed to pay humans for completing the HIP.

5.4 Conclusion

So far, there is no silver bullet for the problem of automatically distinguishing between humans and machines. The original Turing Test describes a human carrying out the test. If this task is delegated to a machine, getting it right becomes much harder. None of the methods described above completely meet all of the requirements for human interactive proofs. One of the biggest problems is that of accessibility. It is absolutely crucial to avoid that certain demographics or human users with disabilities are refused access to services based on a flawed HIP. Another problem is that of resistance against attacks. The current situation with conventional CAPTCHAs resembles an arms race, like the one between cryptographers and cryptanalysts. The harder to break conventional CAPTCHAs become, the less human-friendly they might become as well. Furthermore these kinds of access control are not suitable for every kind of service that is to be protected. The more valuable the abuse of a service is to an attacker, the more likely it is that the attacker will be very ressourceful. This can be taken as far as having an attacker who does not even bother with computerized attacks, but simply pays humans to solve the HIPs for him.

New methods are being developed and there most definitely is a need for them. “Fuzzy” techniques, such as collaborative filtering or implicit CAPTCHAs are an important addition to the toolbox, but they might be only useful in special cases. For every application the right tradeoff has to be found between security against attacks, human-friendliness and accessibility. For example, an online bank needs very strong security, but on the other hand denying blind users access to their finances is not acceptable. On the other side of the spectrum are websites like community forums or blogs, that try to limit the amount of comment spam and similar annoyances. For these applications, weaker forms of access protection might very well suffice. Methods that create accessibility concerns should only be put into effect as a last resort, maybe temporarily during the course of an ongoing attack. If, for example, the collaborative filtering system on a movie community website fails due to pollution of the dataset, a visual text-based CAPTCHA could be used for the time it takes to restore the functionality of the collaborative filter.

Chapter 6

Watermarking

Written by *Newres al Haider.*

One of the big issues in computer science today is the question of how ownership rights of digital media can be protected and enforced. Companies that want to protect their intellectual property often use various copy- and copyright protection methods to try to prevent illegal copying and other tampering of their products. One of the ways to implement copyright protection is to add a certain piece of information to the digital media in question to prove one's rights to it: a watermark.

This chapter gives describes the process of watermarking in digital media, most notably on digital text and images. First a definition of watermarking and it's (desired) properties is given. Next a general watermarking algorithm called Direct-sequence spread spectrum watermarking (DSSS) is shown. Finally it concludes with how this and other watermarking algorithms can be applied to texts and images.

6.1 Definitions of watermarking

While the precise definition of watermarking seems to differ with each author, because they often highlight different aspects of it, the following would be a good summary of it:

Definition 6.1 *Watermarking is embedding a piece of information into a digital object with the purpose of being able to prove a certain property, such as proof of copyright, with it.*

In this chapter the term media or document is used for the text, image or video in which the watermark is embedded. The properties and the definition of watermarking are derived from [HG98], [SHG98] and [PLB$^+$05].

Watermarking is related to two fields in computer science: cryptography and steganography.

Cryptography is the study of securing information. With watermarking it is important to take into consideration what type of media we are watermarking. With cryptography the form of the resulting encryption does not matter while in this case the media should be usable after adding the watermark. This implies that there are more restrictions on how the embedding of the (copyright) message can be achieved.

Steganography is about adding hidden messages to medium. Like with watermarking it is important to keep in mind the format of the medium, in order not to give away that a hidden piece of information is there. Just like with watermarks the removal and or modification of the hidden message must be unreasonably hard without making the original media unusable. It is different from watermarking in that iss main goal is to keep the message hidden. It tries to prevent that the presence of the message is detected in the first place, and if it is detected it wants to prevent the extraction of the message for those whom it is not intended. While this is often same for a watermarking algorithm, and often a steganography algorithm is used for watermarking, the fact that the watermark is detected is not a problem as long that does not imply that it can be removed and/or modified. Another important aspect is that the cover media is unimportant in steganography: any text document, image, video will do as long as it looks like an original without the hidden message. With watermarking the cover media is important and it can not be replaced by another syntactically correct one.

6.1.1 Properties of watermarks

There are certain properties that a watermarking algorithm should or might have depending on the environment and circumstances in which it would be used, and algorithms can be compared on the these qualities:

Robustness It should not be possible to destroy the watermark without damaging the document in which it is contained beyond a certain threshold. The threshold depends on various factors, among them the tolerance for the end user for bad quality media. This should be not underestimated, as some people are comfortable with lower quality media as long as it's viewable/listanable/etc. . Of course it is hard to give an objective measurement for this.

Imperceptibility The watermark should be at best invisible to the end user or if that is not possible it should not hinder the him or her very much, although the first option is prefered.

Security Unauthorized people should not be able to notice, read, remove or modify the watermark. These actions are listed in order of severity as being able to remove the watermark is a bigger problem than just being able to read it. There can be watermarking algorithms were the first few actions are not considered a problem, but an algorithm should always at least protect against the last two.

Fast embedding and/or retrieval Depending on the application of the watermarking algorithm it can be necessary for it to be fast enough to embed "on the fly" especially when the document is watermarked at the time of retrieval from a web server for example. This is especially true for video files where watermarking tends to be time consuming. The retrieval of the watermark does not have to be this fast because that is only needed when a copyright dispute actually occurs.

No reference to the original document In some cases the watermark should be recovered without having access to the original unmarked document. This is because the existence of an unmarked document adds a security risk: if it is copied by an attacker that he/she has in possession an unwatermarked copy. If a watermark was reused than with the original and the watermarked copy he can find the watermark, and could remove it from other documents.

Complex domain processing It is often the case that the media is stored in a compressed form or file format. In this case it would be useful to insert the watermark directly into the compressed media instead of first decompressing it. This is especially true of video because the time it takes to de- and re-encode it.

Interoperability While the watermark is often inserted on compressed media, for some application it is useful to be able to watermark the original uncompressed version when needed.

Multiple watermarks There can be situations where multiple watermarks on document are desirable. In cases where this is not allowed the creator of the algorithm must consider the situation where a watermarked document is watermarked again by a malicious party, which can be a form of an attack as it could destroy the first watermark.

Unambiguity A watermark must convey unambiguous information about the copyright holders so that the copyright can be proved.

Constant bit-rate Watermarking should not increase the bit-rate for applications where a constant level bit-rate is necessary.

6.1.2 Watermerking operations

Digital watermarking can be described as follows: An authorized party (the creator of the watermark and most likely owner of the copyright) wishes to transmit a digital message M (containing for example copyright information, timestamps, etc) through a hostile and noisy communications channel. This is done by embedding the message M with some secret key K into a (cover) media C (for example an image, text or video). This is done by an embedding function Emb(K, M, C). Because the watermark should be invisible to the end user, or at least something very close to it, the distortion caused by embedding must not be larger than a certain value, which is the embedding constraint EC.

The resulting watermarked medium S is then sent over a “channel”, where one must assume that some attacks and tampering will occur by a hostile entity. There are a couple of different ways in which this entity can attack, but in order to succeed he must not harm the cover image with his or her attacks because in this case, unlike with steganography, the cover message is important. This is called the cover constraint CC .

The watermarked medium should then arrive at the destination/end user in such a way that the encoded message M is intact so that it can be recovered with the recovery function Rec(K, S). The key here is most likely the same key as that was used with the encoding, as most watermarking algorithms are symmetric, although there are exceptions. While in this case the cover media C is not used to decode, some algorithms require it, although this is not recommended as it requires and reveals the original not watermarked media. Also it is very important to note that while C is called the cover media, from Bob's viewpoint, receiving it intact is the goal of the protocol. The actual decoding the message from the cover medium is only done if the message must be retrieved, for example if a copyright dispute has occurred.

6.1.3 Attacks on watermarks

There are different kind of attacks that can be made on a watermarked medium. While the precise way of attacks depend on the cover media, they can be divided into these categories:

Erasure and alteration attack In these type of attacks the attacker tries to locate the watermark and attempts to erase or modify it in such a way that the retrieval of the original message will not succeed. Sometimes an attempt is made to change the watermark in such a way that the message indicates a different copyright holder although this is considerably harder to accomplish.

Jamming attack The attacker tries to make changes to alter the media so the watermark can not be reliably detected.

Protocol attack Attacks of these kind rely on some faults in the general protocol around the actual use and application of the watermark. For example if a certain media is distributed many times with a different watermark, this tends to be the case if the watermark contains the information of the buyer of the media, than an attacker could intercept many watermarked copies and by comparing them could figure out where the watermark is located and then remove it. Another example would be if the used algorithm allows multiple watermarks. In that case an attacker could watermark an already watermarked medium and then there would be no way do determine who watermarked it originally as the medium would contain both messages.

What forms an actual attack can take depends on both the cover medium and the type of algorithm used. A good general way is to add additional (white) noise to the media, in the hope that it damages the watermark. Of course this attack must not damage the media too much because it would make it useless for the viewer/user of the media. Because this and related attacks are very common the cover constraint criteria is often defined with the algorithm to show how robust it is.

6.2 Direct-sequence spread spectrum watermarking

In order to give a good definition of direct-sequence spread spectrum (DSSS) watermarking, the definition of what spread spectrum communication (spread spectrum or SS) is needed. The following is a good standard definition of the latter [PSM82]:

Definition 6.2 *Spread spectrum is a means of transmission in which the signal occupies a bandwidth in excess of the minimum necessary to send the information; the band spread is accomplished by a code which is independent of the data, and a synchronized reception with the code at the receiver is used for de- spreading and subsequent data recovery.*

This type of communication relies on the Shannon and Hartley channel-capacity theorem:

Theorem 6.3 $C = B\log(1 + S/N)$ *where* C *= the maximum channel capacity in bits per second with a zero or near zero error rate.* B *= the required bandwidth of the channel.* S/N *= the signal to noise ratio*

This theorem describes the relationship between the maximum information transferred without error per second (C), the minimum bandwidth needed (B) and the ratio of the transmitted signal power(S) and the noise on the channel (S). The interesting aspect of this theorem is that even if the signal power is smaller the noise ($S/N < 1$) it can be transmitted perfectly as long as the bandwidth is sufficiently large.

This applied to watermarking means that watermarking information, which is relatively small in bandwidth when compared to media, is embedded and spread in such way that only the authorized people (those with a retrieval key) can retrieve it.

There are various ways to implement such a watermarking system. In the following subsection I will show one of these possible implementations which is originally presented in [SHG98] .

6.2.1 Definition of a DSSS algorithm

First we take a digital message M which consists of a sequence of bits $b_0, b_1, b_2, ...$ where each bit can be either +1 or -1.

The embedding function Emb is defined as follows: Every bit b_i of the message is repeated N times (where N is the so called chipping rate') which results in the redundant message $m[n]$. With N=3 for example, this produces the following sequence: $b_0, b_0, b_0, b_1, b_1, b_1, b_2, b_2, b_2, ...$.

Next the message $m[n]$ is modulated by a spreading sequence $c[n]$ and then scaled by a factor of $\sqrt{E/N}$. This results in the (to be embedded) watermark $s[n] = \sqrt{E/N} * m[n] * c[n]$. The letter E denotes the embedding power/energy which shall be discussed later in this section. The rest of the embedding function depends on the type of media: for example in text $s[n]$ is used to alter word or line spacings, serifs (the structural details at the end of strokes in letters). With images $s[n]$ can be added to pixel value or to the transform domain depending on what kind of format the images are in. With videos this can happen on the raw image files but it is often the most efficient to use it on the compressed bit stream.

The spreading sequence $c[n]$, which is also known as the carrier, is a pseudo-noise signal.

Definition 6.4 *A pseudo-random noise signal (PNS) are binary sequences which appears to be random, ie. their statistical behavior is the same as for random noise. Formally it can be defined in this case as satisfying the following equations (Note that it does not need to be binary valued as there can be Gaussian pseudo random sequences).*

Binary valued $c[n] \in \{+1, -1\}$, *for any n;*

Periodicity $\sum_{n=0}^{N-1} c\,[n]\,c\,[n + Ni]$, *for any i;*

Zero mean $\frac{1}{N}\sum_{n=0}^{N-1} c\,[n] = 0$*;*

Periodic autocorrelation $\frac{1}{N}\sum_{n=0}^{N-1} c\,[n]\,c\,[n + k] = \delta\,[k]$, *for* $0 \leq k \leq N - 1$*;*

This pseudo-noise signal is usually generated with a linear feedback shift register (the description of these is outside the scope of this chapter) and can be easily (re-)constructed with the proper key K, but given a short segment of $c[n]$ it is very hard to reconstruct the complete sequence. This works essentially the same way as discrete logarithm does in cryptography functions, as it ensures that those without the proper key can not reconstruct this, as it will be shown, is necessary to recover the watermark.

After embedding $s[n]$ passes through the “channel” where one must assume there is some type of interference. These interference are most likely caused by attacks but can also be not maliciously intended operations such as encoding to another file format with images (GIF, JPEG) and with videos (MPEG) or printing with text.

The interference itself is modeled as additive white Gaussian noise (AWGN) $v[n]$ with variance σ_v^2/N. AWGN is defined as a statistically random ”noise” which is normally distributed, continuous and uniform over a certain frequency band.

With this inference the signal which arrives at the receiver’s end as $v[n]$ which essentially correlates to the cover constraint because one assumes that this noise approximates the attack. In addition it also denotes the embedding constraint as the damage to the media when embedding the watermark is represented as this. The resulting message which arrives to the end user with the media is $v(n) = r[n] + s[n]$.

The recovery function Rec(K, S) works as follows: Given a certain key K the spreading sequence $c[n]$ can be generated, in the same way as it was originally generated by the embedder. Than the correlation is measured between $c[n]$ and $v[n]$. This is done by the following equation $\rho_i = Eb_i + \sqrt{E/N}\sum_{n=0}^{N-1} v\,[n]\,c\,[n]$ where the result defines the value of the i-th bit. If $\rho_i \geq 0$ than $b_{(i)} = +1$ else $b_{(i)} = +1$. Using this the whole original message can be reconstructed.

6.2.2 Evaluation of the DSSS algorithm

There are two main values with which the effectiveness of this algorithm is measured:

Signal to noise ratio (SNR) $SNR_{std} = E/\sigma_v^2$

Probability of error (Pr_E) $Pr_{E,STD} = Q\left(\sqrt{E/\sigma_v^2}\right)$ where $Q(x) = \frac{1}{2\pi}\int_x^\infty e^{-y^2/2}dy$

Using these we can measure how well this algorithm fulfills the required properties at the start of this chapter:

Robustness The attacker does not know which parts of the watermarked were altered using $s[n]$ and neither does he know the value of $c[n]$. So in order to perform a jamming attack he/she needs to jam every element of the marked media. Of course the jamming attack must not be too strong otherwise the media will become too damaged to be of any use.

Against a limited power jamming this algorithm affords some protection. This is called the processing gain which is $G_p = N$. This influences the performance in the following way: $Pr_{E,STD} = Q\left(G_p\sqrt{E/\sigma_v^2}\right)$ and $SNR_{std} = G_p E/\sigma_v^2$. If this N is large enough such attacks can be prevented.

Informally the reason for this is that with a large N, the amount of copies of bit b_i which are embedded is also large. In order to make sure that the information in those bits can not be recovered a large amount of these embedded bits must be damaged, which with a big enough N is hard.

Imperceptibility Because $s[n] = \sqrt{E/N} * m[n] * c[n]$ by choosing the N as large as possible we can make the signal to noise ratio smaller than 1. This means that the signal is smaller than the actual standard (white) noise level of the channel (for example the normal amount of noise that is in an audio or video file). This makes it impossible to notice the watermark as the observer does not notice any difference between the watermarked and the non-watermarked media.

Security Without knowing the key K, and therefore $c[n]$, $s[n]$ behaves like standard white noise in the document, therefore it is hard to locate even parts of it. Because c[n] is pseudo random even if the attacker succeeds in getting a certain part of it, it is also hard to retrieve the entire spreading sequence needed to retrieve the message.

Fast embedding and/or retrieval The embedding speed is reasonable, as it takes about $O(M)$ steps to generate $s[n]$ where M is the amount of bits in $m[n]$ assuming that that the sequence $c[n]$ is already generated, and the modulation operation happens in constant time.

No reference to the original document There is no need to use the original media when retrieving the watermark.

Complex domain processing The algorithm given is general: it can be applied to any domain as long as there is enough "space" in which the stream $s[n]$ can be embedded.

Interoperability This is very much depends on the media in question, but it has been shown that the embedding is quite robust on images as the message can be retrieved from an image that was first watermarked and then compressed with a JPEG compression with a quality factor of 20%.

Multiple watermarks It is possible to add multiple watermarks to an image that can be retrieved separately as long as the pseudo-noise sequences used are mutually orthogonal: $\sum_{n=0}^{N-1} c_k\,[n]\,c_l\,[n] = 0$, for $k \neq l$.

Unambiguity Only those with knowledge of the key K can retrieve the message M, and assuming that the key is kept safe this can be a proof of ownership. On a protocol level additional proof can be added by making some part of the message M public or deposit it with a secure trusted party, so that when the message is retrieved it shows who created and embedded the message and because of that who has the copyright.

Constant bit-rate Watermarking in the spatial domain, for example at pixel level, should not affect the bit rate. However one must be careful if the media is compressed in some way, for example if compression is part of the file format like in MPEG. If there are a lot of values that with compression can take up very small amount of "space", for example a high number of consecutive 0's, than if these values are changed than compression is suddenly not as effective as before, so the media shall be significantly larger after watermarking.

As it is shown the described watermarking algorithm is a good, general purpose watermarking method that can be applied to a variety of media. One big issue that has to be taken into account is the choice of the signal power E and the chip rate N, and how they relate to each other: $|s[n]| = \sqrt{E/N}$.

A high signal power will result in a stronger signal, so it will be much easier to retrieve the message correctly even after jamming attacks. However if the signal power is too large than it will have a signal strength in excess of the expected white noise level which will cause noticeable distortion. If the spreading sequence is not chosen correctly this would allow the attacker to easier find the correct sequence in the media, which could lead to a successful watermark removal attack.

A high chip-rate would make sure that the signal strength is spread over a larger part of the available bandwidth and puts $s[n]$, if correctly chosen relative to the signal power, under the white noise level for the channel making it hard to detect. Because with a high chip-rate more bits are modulated for each bit $b[i]$, the ability to resist jamming attacks is increased. However there is a limit on how big the chip-rate can be as it is multiplied by the message length and than has to fit in some domain of the media. For long messages and high N there might not be enough room to embed the whole sequence. So depending on which type of media is used and the length of the message, there is an upper bound on the chipping rate N.

So in order for the algorithm to work effectively the signal power E, chipping rate N and the length of the message must be balanced in regards to each other and the type (and format) of the media used.

In the following section, it will be shown how this algorithm applies to different media types, as well as the main issues when watermarking those type of media in general.

6.3 Watermarking text

While with digital watermarking most people think about protecting multimedia, text documents are often more valuable than video or images, for example: contracts. A big issue with digital texts is that they are possibly hardest to secure as often small parts of the watermarked text are copied. This, among other reasons, made publishers of texts vary of putting them in digital form. There are a number of ways in which watermarking of texts can be implemented as shown [KH02] :

Line shift watermarking This method uses the shifting of lines in a document to hold the watermark. Each line's relative position to the usual represents one bit of watermark information. The encoding done by moving the base line shifts closer or further to each other or the margins the previous depending on the bit it represents and with decoding these relative changes are measured.

Word shift watermarking The watermark is encoded in the relative distance of the words to each other. A word is shifted closer or further the next one horizontally to encode a bit. Some lines are left unchanged as a reference to use when decoding. The main advantage of this system over line shift marking is that there is more space to encode the watermark in.

Feature watermarking In this case the watermark is encoded in certain features in the text such as the serifs. Serifs are the structural details on strokes in a character. Depending on the bit one letter could be a serif or sans-serif (without serif). This gives even more space to encode in than word shift encoding as there are more characters than words in a document.

Inter-character space watermarking The watermark is in this case is encoded in the space between characters. This is used in languages such as Thai where normally there is no or few inter-word spaces, because otherwise spaces between words could be a problem for the algorithm. This drawback makes it unsuited for general document watermarking.

High resolution watermarking The watermarked document consist of two or more components where one is a watermarked object which contains embedded binary symbols. That object resolution is high enough that the human eye can not recognize the binary symbols yet it is robust enough to withstand printing, faxing, etc without loss of information. The pattern of the binary symbols can still be recognized by a computer and can be converted back into a message.

Selective modifications of the document text There is a master copy made of the document and than in the distributed document selective changes are made by replacing some words in a way that the meaning of the document does not change.

> The main advantage is that this works even on document formats such as ASCII, where no additional formatting is used and therefore there is not enough "formatting space" in which the usual watermarks can be embedded. The downside that it needs a master copy to decode the information, and it adds another layer of problems to deal with, for example these documents must be kept safe and secure in addition to the key.

A common way to attack watermarking in texts is to print it and scan it a couple of times or fax it. This way the document remains readable but the finer details where watermarking information is often held is lost. Many of the methods of encoding work only in formatted text as there is no such information in a ASCII text. This implies that a way to attack these watermarks is to change it in a ASCII document and then possible remake it into a formatted one although this takes much more time and effort than the standard print and scan attack. Another important issue is how much space does an algorithm needs for watermarking. This is especially true for a system such as DSSS where the effectiveness relies on high chip rate, which makes the number of embedded bits very large (or at least as large as possible). Methods such as watermarking in serifs are preferred (in theorry) as there is more room to embed the watermark. Robustness is also very important as often only parts of the document is copied, which can make non-robust watermarks unusable.

The DSSS algorithm as described in the previous section was also implemented using line shifts and word shifts. With the former even after 10 times photocopying the watermark message could be extracted, while with the later 75% of the word shifts were detectable on a page after 4 times of photocopying.

6.4 Watermarking images

Digital images are a very common form of digital media where watermarking is used. There are however a number of issues that must be taken into account when watermarking is to be used in these cases.

Images are usually not stored as raw image data but in some kind of image format to reduce the image size considerably, such as JPEG. Because of this, watermarking must be resistant to encoding the image into another format, and/or if that is not possibly the embedding must be done in the domain what the format uses instead of the pixel domain, which is in the case of JPEG the transform domain. If only the later is done however the watermark can easily be successfully attacked by converting it into another file format.

The DSSS algorithm as described was used to watermark images with the results described in [SHG98]. The watermark is not visible in the image. It was tested against an additive noise attack (in which noise is added to make it impossible to retrieve the watermark) and against JPEG compression with a quality factor of 20%. The watermark survived in both cases.

One big issue with images is that they are often scaled, cropped, rotated and otherwise edited. Against such attacks most algorithms are not very robust (including the

On the left the original image, on the right the watermarked image.

Figure 6.1: DSSS applied on an image

one given). A good way to eliminate these problems is the use of templates [HR00] . It proposes that embedded messages should be placed in a certain pattern, in this case a circular lattice with random angular displacement. These patterns are called templates, and in addition to the key these should be known when retrieving the message. If one makes it a requirement that are cyclic shifts in the pattern are unique, by making sure

On the left the effects of an attack with additive noise, on the right the result of a JPEG compression (13.2:1 compression). The message was recovered in both cases.

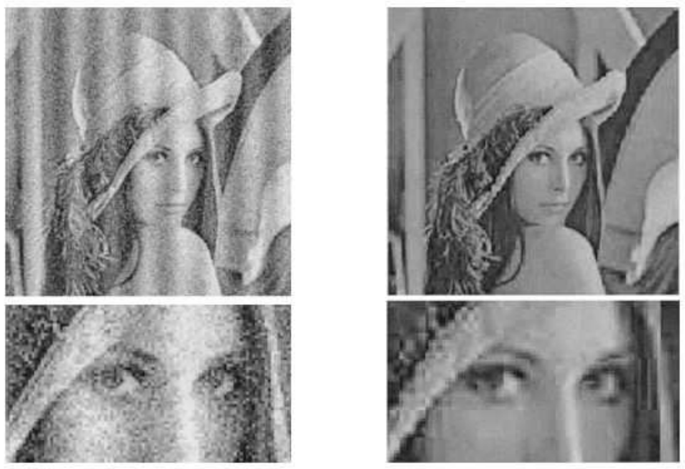


Figure 6.2: The effects of an attack

that the autocorrelation is a delta function (in a similar way as that is the case with the spreading sequences), one can find the new pattern after rotation and scaling. This is done by taking the original pattern and try to scale and rotate it until using that pattern a correlation is found like normally would happen with the algorithm. Because this correlation is unique, than the image and the results can be adjusted in such a way that retrieval of the message will succeed.

6.5 Conclusion

As it is shown the watermarking algorithms used now are reasonably strong against attacks, however there are many improvements possible to improve the robustness and the general usefulness of these.

One area that is needs to be addressed are the protocols surrounding the application of the algorithms. While this is often skipped by the author of the algorithms, in order to make the reliable use of such algorithms possible these issues must be taken into account. For example the problem that an attacker watermarks an already watermarked medium and claims it as its own. Or how the actual retrieval should take place as one must reveal his or her key to settle copyright disputes and if that key is known than all watermarks made with that key can be removed by an attacker. A solution to the later are asymmetric watermarking algorithms, where there is a different need for encryption and decryption.

One application of watermarking that was not mentioned in this chapter (aside from other multimedia such as images and video) is the software watermarking. Just like text and multimedia is copyright issues are a problem with software too, and its domain is very much different, which makes it an interesting topic for research.

In summary digital watermarking is a quickly developing domain, which is closely related to cryptography and steganography, that has much needed applications, especially when it comes to copyright protection of digital information.

Chapter 7

Social Engineering: the weakest link

Written by *Rob Verkuylen.*

The hack into a system is usually done through the path of least resistance. Current cryptographic systems by themselves are so secure and virtually infeasible to break into. The weakest links have become the users of the system. Social engineering is an interesting topic based on the studies of social psychology. A lot of information can be found on either of the topics, but this paper tries to combine these. Although in practice social engineering is certainly feasible, it cannot be scientifically founded and success is not guaranteed. Preventing social engineering can be done by creating awareness through education.

7.1 Introduction

Security is not something you can buy. It is a product, management dedication, company policy and requires dedicated and well educated users. Statistics of a study by the Computer Security Institute [CSI05] show that security is definitely not top priority in most companies and security issues are mostly solved by installing a better firewall, anti virus and intrusion detection systems. Something you can buy, install, and move on.

A popular saying is that most companies spend more money on coffee than on security. One could say that at least people will be alert, but the effect wears off and security is a round the clock matter, not 30 minutes.

The human factor in security is the most important one. Cryptography systems have come a long way since Julian Caesar's attempts at cryptography. The problem this paper will address is that all these systems are only as good as their weakest links, us. We will attempt to look at humans as if they are a cryptographic system. We will look at it's design, how certain input leads to output and what the weak points of this system are. Social engineering[1], hereafter referred to as SE , exploits these characteristics of

[1]I will refer to Social Engineering as SE, in the remainder of this paper. It's a term often used and it will save much trees and your concentration if I do so. Apologies for any offended readers.

the human psyche, just like any other flaw in a security system can be exploited.

First we will take a look at SE itself, how it is used and why any serious security engineer takes this problem just as serious as the key length used in his cryptographic algorithms. In section 7.3 we will take a look at the underlying designs of the human psyche witch allow these 'mind-hacks' to be a success. This area is vast in its own right and has many relations to Social Psychology which describes the way humans tend to act in a social environment. Finally we will take a look at the way we can 'patch' ourselves against these attacks and hopefully create awareness for the readers.

7.2 Security engineering

Security engineering is the field of engineering dealing with the security and integrity of real-world systems. It consist of three main area's: Physical Security, Information Security and Computer Security.

Physical security describes measures that prevent or deter attackers from accessing a facility, resource, or information stored on physical media. It can be as simple as a locked door or as elaborate as multiple layers of armed guard posts. *Information Security* is the protection of information systems against unauthorized access to or modification of information, whether in storage, processing or transit, and against the denial of service to authorized users or the provision of service to unauthorized users, including those measures necessary to detect, document, and counter such threats [NST00]. *Computer security* is a field of computer science concerned with the control of risks related to computer use. Objective is to attempt to create a secure computing platform, designed so that agents (users or programs) can only perform actions that have been allowed.

SE poses a threat to all these fields of security engineering. You can have a 20 ton vault door, with 30 Swiss locks, but if you leave the key chain in your car a car thief can open it. You can have great password security, but if your user finds the password to difficult, because he has to change it every month, minimum length of 8 characters and have capitals, digits and a special character in it, he will write it on a post-it or somewhere in his agenda. Even public-key cryptography, nowadays seen as one of the most secure means of cryptography can be compromised by a user clicking on the wrong attachment on his computer.

7.2.1 Communication

Everybody is a Social Engineer. If you communicate with people then you are a social engineer. You are using your social skills to persuade another person to do something. Getting to like you, going to that movie tonight. Parents try to persuade their children to behave, clean up after themselves, be courteous to other people, don't eat to much candy etcetera. Teachers are persuaders too, they try to get you to find the topic they teach interesting and stimulate you to invest time in the topic and even pass the exam. SE is such an intricate part of our everyday lives that it is hard to pinpoint when exactly someone is doing a bad thing. Salesmen and commercials are trying to get you to buy their products.

Using SE to hack[2] your way into a computer system and retrieve or manipulate confidential information requires a lot more then just persuading someone into doing a single act.

The most famous social engineer, Kevin Mitnick [MS02] has also written a book which gives us some inside information on the workings of a hacker at work. He used a combination of his social- and computer hacking skills. You cannot hack into a machine without knowledge of that machine. Social skills are one thing, but persuading someone into doing the hacking for you is another. This paper will focus on the social persuasion part of SE. Often the true hack into the computer system is the easy part, getting the appropriate authority to do it, the hard part. Although not all social engineering attacks require you to hack into a computer. A common denominator is that the SE gathers or manipulates information. Most of the information we use today is stored on a computer somewhere, therefore access to that information is often through a computer system. These systems are operated by humans and thus a good place to start if you need access is that computer operator.

A complete study of people in social behavior exists in the field of social psychology, the research for this paper required reading of a lot of SE scenario's which described how a SE hack went down. Most if not all the tactics that were given came directly from social psychology with a twist so they could be used for this specific application. Psychotherapists and social engineers are very much alike, they both try to persuade a person into a change of behavior. Though a psychotherapist is most likely interested in a permanent change in behavior and as we will see this will take a lot of time, social engineers are renowned in their patience. Preparation times of up to 6 months are not uncommon. They can work with the target, the person they desire change in, for long periods of time. Gradually changing attitudes and related behaviors to get them into position to execute the hack. There are a lot of short-term ways to get a person to do something they would not do if they were really thinking about it, catching them off guard in their heuristic mode of thinking. The next section will describe some of the main theories of social psychology and how they are applied to persuasion and change.

7.2.2 Scenario's

It is impossible to write about SE and not give the reader some examples, because as we will learn, examples are a very powerful means of persuasion. A lot of scenario's are found on the internet, most not real, but helpful in showing the general concepts of SE.

Although social engineering is much more than getting someone to give their password, some even call it an art-form. Performing an art implies a special skill of the user and since we are all social engineers in one way or the other as discussed in the previous paragraph, I will not refer to it as an art form. The following example will be relevant to the topic at hand, which is security. It is selected from Mitnick's book and is under the topic "Let me help you". Social engineers mostly pick out people with limited

[2]Hacking is a much debated term, some say cracking is more applicable to malicious attempts of hacking. I acknowledges this, but for uniformity I will refer to it as hacking throughout this document.

knowledge of computers, people he refers to as 'computer-challenged'. They are most likely to work with you if you impose to be somebody of authority. This example makes use of new employees. They are very prone to an attack, because they don't know many people in the company, they are not familiar with the company policies and procedures. Mostly they are eager to be cooperative and quick to respond in order to make a good impression with their colleagues and manager:

Call 1:

- Human Resources, Andrea Calhoun
- Andrea, hi this is Alex, with Corporate Security
- Yes?
- How are you doing today?
- Okay, what can I help you with?
- Listen. We're developing a security seminar for new employees and we need to round up some people to try it out on. I want to get the name and phone number of the new employees of the last month. Can you help me with that?
- I won't be able to get it 'til this afternoon. Is that okay? What is your extension?
- Sure okay. It's 52 oh uh, but I'll be in meetings most of today. I'll call you when I'm back in my office, probably after four.

When Alex called at 4:30. Andrea had the list ready, and read him the names and extensions. Rosemary Morgan is one of the new employees from the list Andrea has so helpfully provided.

Call 2:

- Hi, this is Rosemary Morgen.
- Hi Rosemary, this is Bill Jorday, with the Information Security Group.
- Yes?
- Has anybody from our department discussed best security practices with you?
- I don't think so.
- Well, let's see. For starters, we don't allow anybody to install software brought in from outside the company. That's because we don't want any liability for unlicensed use of software. And to avoid any problems with software that might have a worm or a virus.
- Okay.
- Are you aware of our email policies.?

- No.
- What's your current email address?
- rosemary@ttrzine.net
- Do you sign in under username Rosemary?
- No it's R-underscore-Morgan.
- Right. We like to make all our employees aware that it can be dangerous to open any email attachment you aren't expecting. Lots of viruses and worms get sent around and they can come in emails that seem to be from people you know. So if you get an email with an attachment, you weren't expecting you should always check to be sure the person listed as sender really did send you the message. You understand?
- Yes, I've heard about that.
- Good. And our policy is that you change your password every ninety days. When did you last change your password?
- I've only been here for three weeks so I'm still using the one I first set.
- Okay that's fine. You can wait the rest of the ninety days. But we need to be sure people are using passwords that aren't too easy to guess. Are you using a password that consists of both letters and numbers?
- No.
- We need to fix that. What password are you using now?
- It's my daughter's name Annette.
- That's really not a secure password. You should never choose a password that's based on family information. Well, let's see you could do the same thing I do. It's okay to use what you are using now as the first part of the password, but each time you change it, add a number of the current month.
- So If I did that now, for March, would I use three, or oh-three?
- That's up to you. Do you need any help on changing the password?
- No thanks, I know how.
- Good. One more important thing we need to talk about. You have anti-virus software installed on your computer and it's important you never disable automatic updates, even if your computer slows down every once in a while. Okay?
- Sure.

- Very good. And do you have our phone number over here, so you can call us if you have any computer problems?

This is just a simple example of a possible scenario of a SE hack. As stated before the targets were prone to attack because they had limited knowledge of computers and were new employees. Don't think that even knowledgeable workers will not do the same things under different circumstances. In my experiences as an IT Consultant I have been given domain administrator and root accounts within the first 4 hours of entering a new company, just because I asked them. I was a consultant, there to do a job, so I was trustworthy. Contrary to what you would think, the administrative accounts were the ones set to a very simple password and set to 'does not expire' just because at least 5 people of the IT department had to know the password and during the years a lot of programs and schedules were hard coded with this password, so they were afraid to change it, because who knows what will stop working.

Another important topic is password policy. You might say that the above scenario does not apply when a company implements a strong password policy. This is not true either. Passwords are made so that people can remember them. In stead of Annette03, they will use '@NN3tt303', which is perfectly secure right? It is 9 characters long, has uppercase and lowercase characters, digits and even a special character. Brute force cracking with tools like Symantec's LOphtCrack will take the internet-lingo into account and when doing a dictionary attack it will try each word normally and try it again in different forms of character replacement, the 'a' with a '@', the 'e' with a 3 etcetera. So basically hacking this password is just a dictionary attack, especially if you know the name of the daughter and know that the last 2 digits are the number of the month.

7.2.3 Requirements

What does it take to be a Social Engineer? Three terms pop up regularly in articles: 1. Patience, 2. Be natural, 3. Great memory. *Patience* is a requirement we hinted to in chapter on communication, changes come best in small doses. Don't rush into a conversation, build-up to what you are getting to. *Be natural*, background research of the target person and or company are very important as we will show in the next chapter. If you imitate to be the cable-guy, you have to know all the lingo used for that job. If they ask you for the TPS report and you don't know what it means, you probably don't get another change.

Social engineers are known to have *good memory.* Mitnick writes that at very young age he knew how to get around for free on every bus in the San Francisco area because he was able to remember the entire bus schedule for that area. In communication with your target you have to remember each and every detail of your background research and be able to apply it. These are the facts and cues you will learn about in the next chapter. They are essential to persuasion.

7.2.4 Background research

A very important part of a social engineer is doing research on the topic he is involved in. This can be information of the company the target works at, his or her colleagues or the target itself. Internet and good search engines are very helpful. Mitnick shows a few examples of complete manuals being available on the web. The Program Operations Manual of the Social Security Service can be found online, explaining in detail the lingo used, which procedures to follow and how to ask for information. Perfect information for a social engineer making a call. Even the FBI has documentation online with instructions to get information from the National Crime Information Center, which procedures to follow and what syntax is used for information retrieval.

A lot of people have personal websites and if they had any life online, chances are you can find at least some information on them. Sites for reversed phone number lookup can give you an easy address and even sites like classmates.com(dutch www.schoolbank.nl) can give you insight in the education and thus possible interests of the target. The organizational diagram of a company is great information for finding ouy who has authority over your target, or what department they work in.

Another well known form of research is called 'dumpster driving', all the trash we throw away leaves a big trail of information of what we do and who we have come into contact with. There is so much information available people put out, thinking that single piece of information is not interesting for a random person. For a person who is really intent on finding out, these pieces of information will help him a great deal to get into contact and make up a credible story, based on these small facts of the target's life.

7.3 The human cryptosystem

Social sciences have been working for over 50 years to study human behavior in a social setting, which for most of us is everyday life. Although we still cannot understand exactly how the cognitive processes of the human brain works, we can experiment with it as if it were a black box. A certain input results in a certain output. Human nature is not something you can make on a circuit board, but statistics can learn a lot about how we will react given certain parameters. Scientists like Cialdini [Cia80] have done extensive research in the field of persuasion and getting cooperation. This chapter will give insight in some of the most important aspects of social psychology, related to persuasion and getting people to change.

7.3.1 Influence, persuasion and attitude

Influence is an umbrella term. Any time a source deliberately attempts to change a receiver's thoughts, feelings, or behaviors, influence has occurred. Persuasion is a special case of influence. When a source deliberately uses communication to try and change a receiver's attitude, then persuasion has occurred. Both influence and persuasion concern deliberate change, but diverge because persuasion requires communication and seeks attitude change. Influence can proceed without communication and may achieve behavior (external) change without gaining attitude (internal) change. One final key word: Attitude. Attitude is a person's evaluation of an object or thought. A person holds up an evaluative measure (good to bad) and judges objects against that scale.

Persuasion is the key for a Social Engineer. Persuasion seeks to change attitude and attitudes drive behavior. Persuasion can also come through communication and that is exactly what we have an abundance of. Methods of communication, methods to reach the target for the attack. Although influence is the most direct way to change a person, it has drawbacks and the effects will only last for a short time. You can threaten someone to do something and if the threat is severe enough the person will most likely do it. But the influence only last as long as someone has that control. It is possible to influence a person to do something, but as soon as the source of influence leaves, the person will revert back to his/her original behavior. Therefore, if you cannot directly control another person's behavior, you have to find a better way for getting that behavior.

Attitudes drive behavior, if you want to change other people's behavior, you need to change their attitudes about the object in question. A classic example is arachnophobia. If you persuade a person that spiders are harmless, though an extensive build-up of exposure to spiders, you will change a subjects attitude toward spiders and therefore their behavior, they are not (as) afraid anymore. The same holds for SE tactics. If you can change the attitude of the target toward you or the people they work with, you can change their behavior.

Certain conditions seem to improve the likelihood that people will show attitude-behavior consistency. If these conditions are not met, research shows, attitudes will not drive behavior. These conditions are called 'Attitude Availability' and 'Attitude Relevance'. An attitude must be available, you need to think about it at the time it is applicable. It can be available, but it also needs to be relevant for the situation at hand. You know you could cheer for the goal of the opposing team, but it's not relevant, so you don't do it.

7.3.2 Dual Process Persuasion

How do people change? The dual process approach claims that a person's mode of thinking determines influence. The dual process approach is based on four assumptions about people and influence.

Assumption 1: There are two distinct modes of thinking that a person may employ. *Systematic-mode.* The systematic mode refers to a person who is carefully and effortfully thinking. The thought process is active, creative, and alert. *Heuristic-mode.* The heuristic mode, by contrast, is at the other extreme of thinking. Here the person is not really thinking very carefully and instead is skimming along the surface of ideas. They are thinking enough to be aware of the situation, but they are not thinking carefully enough to catch flaws, errors, and inconsistencies.

Assumption 2: Situational and personality variables affect which mode of thinking a person will employ. People are flexible in their thinking and can move back and forth between the two modes. Sometimes we are systematic and other times we are heuristic. The mode we use depends on situational and personality factors. This happens all the time when you read a newspaper. Reading about a new scientific breakthrough which interest you very much, will set you in systematic mode, wondering how and why etc. Skimming through a gossip magazine you are not looking for facts, but more for the feeling these give you. People also have strong individual preferences for particular modes of thinking. Some people have a high need for cognition and typically think carefully about things most of the time. By contrast some people have a low need for cognition and typically think as little as possible about a situation. In between are most people who are more sensitive to situational factors. There are several factors that can give a clue of the most likely preference of a person. Education, age, type of activity of the person is working on at the moment all give hints. If a person has been doing the same desk job for 10 years, they are most likely on 'auto-pilot' and don't need a lot of cognition to do their job. If a system administrator is just fixing that server-crash of last night, you are certain he is in systematic-mode.

Assumption 3: Persuasion variables will have different effects depending upon the mode of thinking employed. When people are in the systematic mode, certain things will be very important and influential to them. When reading the scientific article people will look for facts, evidence, examples, logic. These are *arguments*. In contrast, when people are in the heuristic mode, other things will be important. Since arguments (facts, evidence, reasoning, etc.) require a lot of cognitive effort and energy, the heuristic thinker won't use them very much. Instead, easier to process information will be employed. Things like the attractiveness, friendliness, or expertise of the source will be more influential for the heuristic thinker. These are called *cues*. Depending on the target's mode of thinking, success varies on the use of arguments or cues.

Assumption 4: Influence achieved through the systematic mode is more persistent over time, more resistant to change, and more predictive of behavior than influence from the heuristic mode. When people are thinking systematically, if they are influenced, it is more likely to stick precisely because they thought about it more carefully, fully, and deeply. For heuristic thinkers, however, any influence is likely to be rather short lived, simply because they did not really think that much. Note that this says nothing of the magnitude of the attitude change, only how long it stays with the target. Most research shows that there are actually only two modes of thinking and nothing in between. People will switch between modes and even apply them at the same topic, but mostly never at the same time. Studies indicate that most people are 'lazy thinkers', most of the time, we are in heuristic mode. Carefully and effort fully thinking means spending a lot of energy and we just can't do that all day every day. Try watching or listening to a block of commercials and see how much thinking you had to do. Commercials are full of cue's and do not have many facts. For SE sometimes you don't want a target to actively think about what they are doing, going over all the consequences, but sometimes you do. There are many ways to switch. Two important criteria of a topic are: *Relevance.* A topic needs to be meaningful and relevant to a person. A teacher needs to show his students a topic is relevant for them in order to get them to think systematically. *Comprehension.* This is very important and often overlooked. Most people don't tell you if they don't understand something because they feel embarrassed. They nod and say they understand, while they don't.. If a topic is too complex for a person they mostly try a few moments to be systematic, but will switch back to heuristic-mode when they don't comprehend. If you need a lasting change, make sure a person understands. In order to persuade someone you need to know what are the arguments or cues that are important to the receiver. A car salesman tries to get the correct arguments all the time. A younger person probably wants a fast compact car. A young parent wants a car that is practical for his kids. Cues. Beer commercials in the USA consist mostly of half naked women around beers. In the Netherlands it consists of commercials showing 'the guys', just some guys getting together in a sometimes funny way. Neither of these trigger you to think about beer, they give you sensory information which the target audience, mostly men, associate with a pleasant feeling, without giving thought to it. The good feeling is then associated with beer.

The most difficult part of persuasion is understanding what is important to a person, so you can apply the correct arguments and cues. Background research is very helpful in this area, if you can find out the likes and dislikes of a person, you are most likely able to come up with some relevant arguments or cues for persuasion for that specific person.

The behavior of people tells you a lot about which mental state they are in. Are they alert, thoughtful or do they look bored, distracted? Check for the response to a question, the answer tells you a lot of how well thought through it is. When people are in systematic mode they look and act differently than when they are in heuristic mode. Reading these verbal and non-verbal signs means you can adapt to a situation and use the right tool for the job, arguments or cues. When in doubt take the peripheral route, since people are in heuristic mode most of the time.

7.3.3 CLARCCS

Cialdini [Cia80] derived six general cues of influence. These cues appear to transcend occupation, region, personality and education. In other words, they work in many different situations. These six cues also share another important similarity, they work best when a person is in heuristic mode.

Comparison. We like to compare our behavior against the standard of what everybody else is doing. If there is a discrepancy, we adjust. Laughing tracks in comedies are a prime example. People laugh more at a comedy with a laugh track, than the same one without. A street musician with his opened guitar box always has some money in it, because we don't mind giving some if others already have, but we don't like to be the first.

Liking. People have the tendency to comply when the person making a request has been able to establish himself as likable, or as having similar interests, beliefs and attitudes as the target person has. This is very important in sales. Service with a smile. If you like someone you will buy easier, more and more often. Attractiveness is also associated with liking.

Authority. When the source is an authority, it must be true. When we hold someone in regard as an authority on a subject or in general, we tent to believe that person. A nice example was Frits Bom, a dutch tv show host who had a good reputation on shows helping people with problems and giving tips on the best holiday locations. When he suddenly appeared as an actor on a commercial, using almost the same setting for selling loans for a loan company. The commercial was banned, not because of the actor using his authority, but because the setting was too much like the original show. Reciprocity. If you receive something, you need to give something in return. When a stranger approaches you and says: “Good day to you sir!,,, you feel compelled to give at least some form of acknowledgement. The source gives you something, now you feel obligated to give something back. If you get invited for dinner at someone, this implies that you ought to invite them at least once. If that person served you up a 5 course dinner and you server them out pizza, they will be angry.

Commitment/Consistency. When you take a stand, you should stick with it. The four walls sales technique: 'Do you think health is important?' 'Yes.' 'Do you think exercise is important for you health?' 'Yes.' 'Do you think food supplements can improve your health?' 'Well yes.' 'Ok I sell vitamin supplements, may I help you improve your health?' This is a very powerful persuasion technique. Another application is 'bait and switch'. You see a great add laptop for just 800 euros, it has everything you always wanted. You feel exited about getting a new laptop with all the features you ever wanted. You race to the shop and tell the clerk to get you one of the laptops on sale. He walks to the storage room, comes back and says he's sorry that that advertisement model is just sold out, but he still has this one that is just 100 euros more but it's just as good or even better. Most people will accept the deal, because they don't want that exited feeling of getting a new laptop to go away and maybe they even bragged to their friends about getting one. In the Netherlands a debate is going on about advertisements of travel agencies. Advertisements say you can travel from Holland to Egypt, for just 199 euros. But these prices exclude a lot of extra charges, like airport taxes, security fees, last minute fees etcetera. Once the customer is in the shop and went through the whole ordeal of selecting a flight and a hotel he is less likely to turn the deal down one he sees he has to pay double the money for the trip.

Scarcity. When something is rare it has high value. If you can get your hands on something that not everybody has, then you feel like you have something special. You were the first to get that new gaming console, so you are willing to spend all you money on it. Next year you can get them half price, probably with all the bugs fixed, but hey, by then everybody has them.

These cues are used as mental shortcuts when we are in heuristic mode. Receivers can easily apply these cues to guide their thinking or action with a minimum of mental effort and activity. Most of the time these are really helpful in everyday life, that is why people trust these cues and don't think about them. When someone goes to the systematic mental state and starts to really think about them they are not rational and even seem ridiculous. All of a sudden they are not effective anymore.

7.3.4 Attribution

No matter the cause, we have a strong need to understand and explain what is going on in our world. Because people must explain, it opens up some interesting influence possibilities. If you can affect how people understand and explain what is going on, you might be able to influence them too.

There is a theory about how people explain things, the Attribution Theory. Whenever we offer explanations about why things happened, we can give one of two types. *External attribution.* An external attribution assigns causality to an outside agent or force. *Internal attribution.* An internal attribution assigns causality to factors within the person. An internal attribution claims that the person was directly responsible for the event external attribution claims it was something outside the sphere of control of the person. We use this to explain every single event in our lives. If you failed an exam, you are most likely to say that the teacher was lousy, the questions were all wrong etcetera, all external factors. If you got the highest grade in the class, it was all you, you were the best. In general, if you can't explain it yourself, you make an external attribution.

How is this related to persuasion: If someone asks: 'Why?' and you provide the correct attribution, their behavior will change according the attribution. If you wonder where lightning came from 3000 years ago and I told you that Thorr, the god of thunder was angry at you, you ran and sacrificed some more grain or sheep. Now if I explain the current day beliefs, you won't go out and play golf anymore when you see a big black thundercloud coming toward you.

The key to change are internal attributions [S74]. It changes attitudes and believes about the person itself. Attribution theory is also important in encouragement and self-esteem. If you encourage someone and compliment their work and say 'You worked really hard on that, good job!' or 'Wow your really good in math.' people will attribute the results to themselves and change their behavior. Next time they face a difficult math problem they will 'know' they are good at math so will not give up as easily.

The problem with external attributions to change is that they are imposed by an external force and will only last as long as that force exists. This problem is found in imprisonment, most prisoners are very social and helpful in their time in prison. But mostly only because they will have a harder time if they don't and might even get a time reduction if they do. As soon as that pressure is gone, ex-inmates revert back to their old behavior. External attributions are not all bad, but are only useful if the person with the attribution links the external attribution to themselves.

There are two key steps to effective use of Attribution. First, it must be applied in a situation where people are thinking about why things are happening. Second, the explanation must be an internal attribution. Most of the time less is more, the less you say or do, the more you let the receiver think about it, the better it works.

In essence attribution theory shows us that people can create new attitudes or beliefs or behaviors depending upon the explanations they make. If they make external attributions 'I'm cleaning up my room, because my wife has been nagging about it the last week', they are most likely not changing their attitudes about cleaning up the room. With an internal attribution 'I'm cleaning up the room, because I like my room to be clean', then their attitudes have changed and effectively their behavior.

7.3.5 Inoculation

Brainwashing, a term introduced when American soldiers were captured in the Korean war and a significant amount of them cooperated willingly with the enemy. This was a great shock to the homeland, and at first the general belief was that a combination of torture and punishment drove these souls to the bad side of the fence. The brainwashing sessions did not necessarily include torture, but rather featured a lengthy debate between the captured soldier and a skillful questioner. And the debate was about America and the American beliefs and values on freedom, democracy and freedom. Many of the soldiers had great difficulty defending their political and social beliefs. They believed that democracy was the best form of government, but they could not explain why this was true. And their captors merely attacked these simply held beliefs until the soldiers began to doubt their validity. After that the road to 'treason' was easy.

To get someone to hold a belief more strongly, you need Inoculation. Inoculation is a medical term, used to indicate that in order to protect a person against a viral disease, you give them a weak version of the virus, so that the immune system can attack it and become stronger. The next time a bigger threat comes along of the same virus, the immune system is ready to defend itself and survive the attack.

The application to persuasion is apparent. If we want to strengthen existing attitudes, beliefs, and behaviors, inoculation theory suggests that we should present a weak attack on those attitudes, beliefs, and behaviors. Again, the key word here is, weak. If the attack is too strong, it will cause the attitude, belief, or behavior to get weaker or even move to the opposite position. The attack has to be strong enough to challenge the defenses of the receiver without overwhelming them. There are 3 steps in inoculation for persuasion.

Warn of the Attack. The warning plays a key role in the inoculation process. It serves to activate the existing defenses in the receivers. As soon as the warning is made, receivers are threatened and get ready for the attack. They start preparing and start thinking about all kinds of scenario's that might happen. Over preparing.

Make a Weak Attack. Now, if you think about it, an attack is simply an act of persuasion. An attack is an attempt by some source to change the thoughts, feelings, or behaviors of receivers.

Make the Receiver Actively Defend. Research have shown that the more actively the receiver defends against the attack, the stronger the existing attitude will become. An active defense occurs when the receiver does more than merely think, but rather performs actions. It may not seem directly applicable, but this is what you do when you go into a meeting, telling everybody you've had enough, or when you prepare for a job application. You don't know what to expect, but you are warned of the 'hard' circumstances, you start over preparing, go through all possible scenario's in your mind and think about what you are going to say. The whole time you're doing this you are toughening up your defenses. The whole point of inoculation is to get people to think for themselves. When people actively generate their own ideas and thoughts, then have to vigorously defend those ideas and thoughts, they will develop considerably stronger attitudes, beliefs, and behaviors.

7.3.6 Sequential Requests

Sequential Request are very often used in sales techniques and are very effective in persuading people to follow through. Sequential requests strategy is simple to implement and effective in outcome. The first step is a set up and is not the true target, rather it is used to get the receiver in the right frame of mind. The second step is the real target. It is the action the requester really wants you to perform.

The first way is called the door-in-the-face or DITF for short. The second way is called the foot-in-the-door or FITD. Both require two steps and do a set up on the first step with the real target on the second step. The difference is how step one hits the receiver. Two major limitations apply to DITF. First, the requests appear to work best if they are social rather than selfish. Second, the requests work best if there is no delay between them.

Principle of DITF: Get a no at large request, get yes at real request. Ask someone if they could do volunteer work for 20 hours a week for a good cause. Most people will say no, because that's just too much work. Then ask them disappointedly if they could 'at least' donate some money to their cause. Human nature is to feel bad about denying at first and will give in to the second smaller request.

Principle of FITD: Get a yes at small request and get a yes at real request. This is very much related to the Consistency/Commitment cue. Ask someone to sign a petition for a good cause, it takes no time and requires almost no effort. Then telling that you are happy they follow your cause and since you are obviously with them on this, can you also donate some money. People feel they need to be consistent, you were agreeing on the good cause, then why not follow through and give some money also. Again the limitations of FITD. FITD works best with social requests just like DITF. Secondly FITD works best when there are no extra incentives offered for performing the requests. You are doing it because of yourself, not because you received money.

7.3.7 Message Characteristics

In this section we will consider a variety of ways in which we can use message characteristics to influence. There are a lot of characteristics, but we will go through some of the most important ones.

Revealing Persuasive Intent. If somebody knows you are going to try to persuade them, they dig in and won't be as susceptible anymore. This is related to inoculation, which we have seen earlier.

Organization. Well organized messages tend to be more persuasive than disorganized messages. If your message is incoherent and confusing, then receivers will have difficulty merely understanding.

Examples vs. Statistics. The research indicates that examples tend to be more powerful and persuasive than statistics. There are several reasons for this. First, examples are easy to comprehend and require less effort. From the chapter on the paths to persuasion, we know that most of the time most people prefer to minimize the amount of thinking they must do. Therefore, examples can function as effective persuasion cues. Examples can be more effective than statistics because people will think more about them. With statistics, about all people do is learn them. If the audience is highly sophisticated and very well informed, statistics would be considerably more effective. In fact, overuse of examples could reduce the credibility of the speaker. Examples and Statistics fall under the category 'Evidence' persuasion with evidence is much stronger than without.

Message Sidedness. There are two sides on every issue. One sided messages discuss only one perspective. Two sided messages present information on both sides. Generally speaking, a two sided message is more persuasive than a one sided message. To be most effective, a two sided message must do two things: Defend One Side, Attack the other side.

Repetition and Redundancy. Repetition is saying the same thing over and over again. Redundancy is saying the same thing in a different way. This is best seen in commercials. Show someone a commercial one time and they will probably forget it. Show it 10 times and they are likely to remember. The problem is, there is a breaking point at which time the repetition becomes annoying. 'Not that commercial again.' The same thing applies to a joke. It's funny the first time, it's even funny the fifth time and by then you can pass it on to your friends, but after 50 times it becomes an nuisance. Redundancy is there to extend repetition. Make a new add every 3 months and you can advertise the same product as long as you want.

Rhetorical Questions. A rhetorical question is an utterance that is really a statement, but looks like a question. Rhetoricals are polite ways of making claims without appearing to take a stand. 'I've got a point here, don't I?' Rhetoricals can be persuasive because they can make receivers think more carefully. If receivers are not thinking very carefully about the persuasive appeal, a rhetorical question jerks their attention and makes them think. The reason for this is due to our social training. When somebody asks us a question, it is required that we respond to it. To respond correctly requires that we understand the question.

7.3.8 Obedience

Obedience is defined as receiver compliance through source authority. The classic example is an officer giving orders to a soldier. The compliance does not occur because the soldier likes the officer or necessarily respects the officer's judgment and expertise. Rather simply, the officer has power and the soldier must obey. Milgram [**?**] has done some of the most interesting and quite literally shocking, research on this topic. To summarize: A random volunteer, the teacher would have no problem punishing a total stranger, the learner, with high electric shocks, just because the scientist says this was necessary for the experiment. Milgram ran this basic experiment with some interesting variations. Sometimes, he separated the teacher from the learner. Sometimes, the teacher and the learner were in the same room, but at different desks many feet apart.

The results: First, no matter how close or how far apart George and the Volunteer were, at least 20% of the teachers would go all the way and deliver the highest most dangerous shocks. In fact, when George was in the other room, over 60% of the teachers went all the way and complied fully with the demands of the study. And of the hundreds of people who participated in this study, fewer than ten refused to participate at all.

There are two general reasons for obedience. The first one is referred to as the 'shallow' reason, the second one, 'deep'. The shallow reason for obedience seems fitting with the dual process theory: People often do not think about what they are doing, they are either systematic thinkers or heuristic thinkers. As we have seen, depending on which state a receiver is in, different variables (arguments and cues) will have very different effects. Finally, we also know that most of the time, most people do not think very thoroughly or carefully about what is going on. Obedience is mentally easy. It is easier to assume that the authority knows what is best and just do what you are told to do. The deep reason for obedience is survival. Humans have been able to survive mostly because we can live in large groups and outsmart bigger, stronger animals. By banding together we can pool our resources and translate our individual abilities into powerful tools and weapons. It becomes imperative, then, that we do what it takes to make groups survive. One of the ways we make groups function effectively is through obedience to the hierarchy of the group. If obedience stops, then the group will eventually fall apart.

In relation to Milgram's research, the now famous Stanford Prison Experiment [ZHBJ73] has become a classic demonstration of situational power to influence individual attitudes, values and behavior. The changes in character of many of the participants were so extreme, swift and unexpected that the planned two-weeks of the experiment had to be terminated by the sixth day.

7.4 Security awareness

The key to protect your assets against a social engineering attack is education. Education to create awareness in all layers of a business. From upper management to the cleaning lady. You cannot expect everybody to know they should not give out their password to somebody who seems to be a legitimate person. Security should become a part of your work, even if you never touch a computer, awareness of the possibilities is important. Gragg [?] proposes a multi layered solution as if digging in a fortress, creating so called Social Engineering Land Mines(SELM's) along the way to detect a SE attack as soon as possible. Mitnick dedicates two whole chapters of his book on this topic alone. We will look into the main topics here and discuss the best practices shortly.

7.4.1 Security policy

Most advisories have the same setup, create a strong security policy as the base of their defense against attacks. Security policies are clear instructions that provide guidelines for employee behavior for safeguarding information. These instructions range from the password policies to policies about opening email attachments.

An important step in creating a good security policy is an assessment of the assets that need protection and what threats could be posed on those assets. A good way to determine how much effort and funding is needed to set up the policy is determining the costs in time and money if one of the threats were to come reality.

7.4.2 Data classification

A data classification policy provides a framework for protecting corporate information by making clear statements about the level of sensitivity of each piece of information. Without these each employee is making this judgment by himself. Mitnick poses 4 levels of sensitivity: Confidential, Private, Internal and Public. Each level has certain procedures to follow and requirements on the user given this information.

7.4.3 Verification and authorization

To maintain effective information security , an employee receiving a request to perform an action or provide sensitive information must positively identify the caller and verify his authority prior to granting his request. This can be done in three steps: 1. *Verification of identity*, this can be achieved in various ways. CallerID, caller has to come from known number, a shared secret, or someone vouching for the caller. 2. *Verification of Employment Status.* The biggest security threat is not from an external hacker, but from a just fired employee who wants to 'set things straight'. Maintaining an active employees list and checking if the employee is still on it is a way to handle this. Asking the callers' manager is two in one with vouching. 3. *Verification of need to know.* Again this can be done through the manager, but also when in a big company with a software package, listing all personnel and all access levels to different types of information. Another way is to check with the information owner if this caller has access.

7.4.4 Departmental policies

Different departments require a unique set of policies. The help desk, HR and receptionists are all people prone to attack and need their own set of rules and guidelines to follow. The help desk worker must never ask for a password. The HR employee must notify the IT department immediately if a person has left the company.

7.4.5 Physical security policies

Though social engineers will rarely show up physically at a site, it is still important to have good policies in place for this area of security. Visitor Identification, every visitor must show a valid ID to be permitted. Escorting, every visitor will be escorted at all times. No visitors are allowed to enter critical area's of the company without supervision. Think about mail- and server rooms. Trash containers never leave the company premises. This prevents dumpster driving. Always shred your papers you will throw away. Maybe even burn it rather than shredding. Special software exists that can scan every little piece separately and put them all together again.

What these policies are trying to achieve is inoculation of everybody in the company. Prepare them with small attacks, get them to think about it so they are prepared for the real thing. Education is the key, with special care that each and every person understands the risks involved if they do not follow these instructions, so they won't take shortcuts. The best way to keep the education at a high standard is repeating courses, just like a CPR course. A reward system for users who have done a good job following procedures, prevented an attack will be an incentive for the personnel to go the extra mile and continue to employ the policies.

7.5 Conclusion

Now that we are at the point in time when cryptography algorithms are so secure that cracking them has become virtually infeasible, it is more and more attractive to look at the whole chain of security and attack the weakest link. People that are not aware of the dangers and who are in a state of cooperativeness can be prime targets of social engineers. We have seen a small example of a social engineering attack, which was only a hint at what is possible and has been done. A great deal of this paper is dedicated to showing some of the weak points in 'the human cryptosystem' in order to create awareness for the reader. The interesting thing about psychology is that it is a topic common to all people and the examples given are most likely familiar to everyone. Once you learn about the underlying theories of our behavior, you can immediately start recognizing them in everyday life adjust your attitude and as a result your behavior.

The key to preventing social engineering attacks is creating awareness though education for all personal involved in managing and using the assets to be secured. Using a data classification system takes away the responsibility at the user level and leverages it to corporate decision making. Together with following a few simple policies to make sure only authorized users get access to information will make the life of a social engineer much harder. I hope this paper has inoculated the reader with insights and some weak attacks that will help him prevent real future attacks on his 'system'.

Chapter 8

Fairplay

Written by *Ewoud Bloemendal.*

This chapter describes the Fairplay project, an implementation of 2-party Secure Function Evaluation.

The description of the Fairplay system given in this paper is mainly based on “Fairplay - A Secure Two-Party Computation System”, by Malkhi et. al [MNPS04].

8.1 Introduction

8.1.1 Secure Function Evaluation

We consider two parties, Alice with secret input x and Bob with secret input y, that want to evaluate a certain function $z = f(x, y)$. To evaluate this function they have to work together, although they don’t want to give the other party more information about their input than the information that can be derived from z. Problems like this generalize many problems known in cryptography, as they can be written as a function with two seperate inputs and a common result. Protocols allowing Bob and Alice to do this are known as Secure Function Evaluation (SFE) protocols. In fact every protocol that solves a problem which can be written as a function $z = f(x_1, x_2, .., x_n)$ where n parties want to know the result z without revealing their input x_i is part of SFE. But as the Fairplay project uses 2-party SFE, we will mainly focus on that version of SFE.

A perfect SFE protocol does exactly the same as a trusted third party, but then without using one! In trusted third party protocols the security lies in the trust in the third party, but in perfect SFE protocols the protocol has to implement this security by itself.

A commonly known example of SFE is introduced by Yao [Yao86], and called the ”Millionaires problem”. Two millionaires called Alice and Bob want to know which of them is the richest, but they don’t want the other to know how rich he or she is. Using a SFE protocol they may find out whether they are the richer or the poorer one. If the protocol tells that Alice is richer than Bob, Bob also know that the amount of money that Alice has bigger is than his own amount, but more information about how much Alice has is not given.

8.1.2 Boolean circuits

Common protocols that solve SFE are using boolean circuits. A protocol that performs SFE on boolean circuits also does this for functions, because each function with a few input bits and one output bit can be written as a boolean circuit.

A boolean circuit consists of wires and gates. Wires are used for input and output of values, while gates are used to perform boolean logic between wires. Gates are nothing more than a truth table with inputwires W_1 till W_n and an enumeration of the outcome on all the permutations of the inputwires. Figure 8.2 gives a visual representation of a gate with two inputwires, also known as a binary gate. Given the values of W_i and W_j the gate will give the outputwire W_k the corresponding value g_{W_i,W_j} .

8.1.3 Fairplay overview

The Fairplay system outputs two java programs, called Alice and Bob, which perform 2-party SFE, following the protocol suggested by Yao [Yao86]. But before the system can output these java programs, Bob and Alice should agree on the function they want to evaluate. To make this possible Malkhi et al. designed a high level procedural definition language called Secure Function Definition Language (SFDL). This SFDL is a C/Pascal type of language that can be used to describe a function. Because the SFE protocol can't use functions, but rather boolean circuit, the Fairplay system also contains a compiler that compiles SFDL into Secure Hardware Definition Language (SHDL). SHDL is a language that describes boolean circuits, as in the example in the previous section. After Bob and Alice agreed on what function to evaluate the system outputs both Java programs, which performs serveral steps to do a Secure Function Evaluation. The complete system including these steps are listed in figure 8.1.

8.2 Definition languages

As stated before, the Fairplay protocol uses boolean circuits to evaluate functions. But to write a certain problem in as a boolean functions is a very difficult task for humans, because boolean circuits are hard to understand. So thats why SFDL was designed, and a compiler was build to compile this language in a usable format for SFE. We will now have a look at first the SFDL, then SHDL and finally at the compiler.

8.2.1 SFDL

The Secure Function Definition Language is a C/Pascal style procedural language. The Millionaires problem can be described in this language like this:

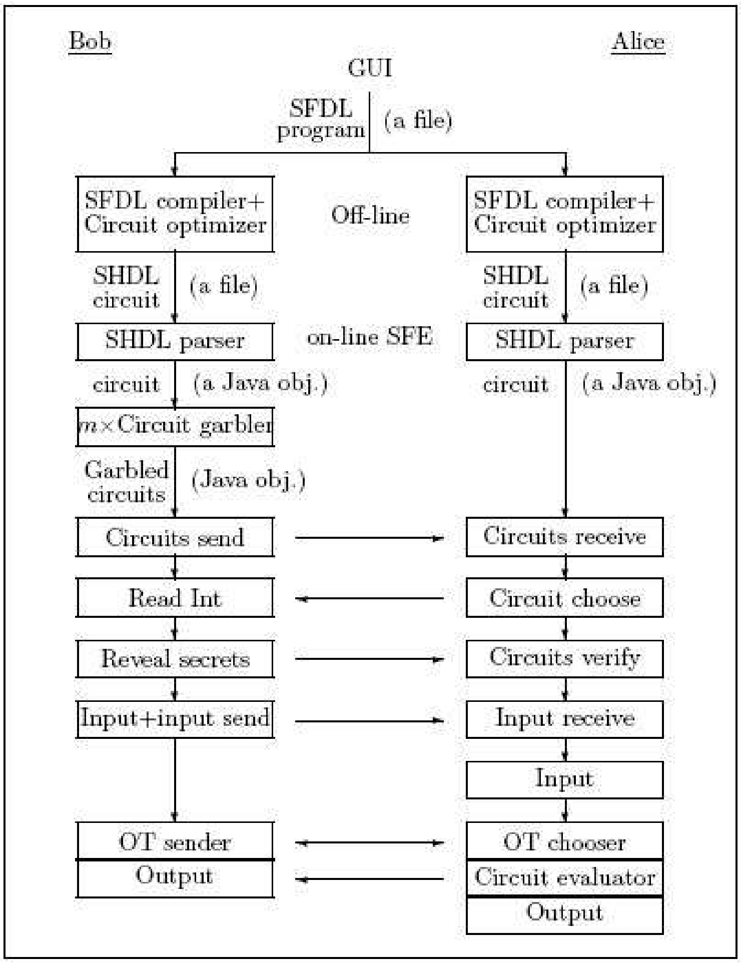


Figure 8.1: Overview of the Fairplay system

```
program Millionaires {

  type int = Int<4>;   // 4-bit integer
  type AliceInput = int;
  type BobInput = int;
  type AliceOutput = Boolean;
  type BobOutput = Boolean;

  type Output = struct {
    AliceOutput alice, BobOutput bob};
  type Input = struct {
    AliceInput alice, BobInput bob};

  function Output out(Input inp) {
    out.alice = inp.alice > inp.bob;
    out.bob = inp.bob > inp.alice;
    }
}
```

The special types AliceInput, BobInput, AliceOutput and BobOutput must be declared in every program. The structures Input and Output are also used in each program. The rest of the program consists of a sequence of functions. The last function is always the function that computes the output, and therefore it is named Output. The SFDL has a couple restrictions. So is recursion or nesting of functions prohibited. All these restrictions are used to maintain the obliviousness of the program.

For the security of SFE it is needed that no party can learn anything about the actual values that are used in the computation. To ensure this obliviousness, SFDL may not contain code that is executed depending on the value of certain variable. So thats why for loops must have compile time fixed number of runs, recursion is not allowed etc. The language does support If Then Else constructions, but the compiler makes sure that both sides of the branch are always computed. Otherwise it would not be oblivious anymore.

For a complete description of the language I refer to the Appendix A of the original paper [MNPS04].

8.2.2 SHDL

Below an example of a boolean circuit, produced by the compiler and described in SHDL:

```
0 input //output$input.bob$0
1 input //output$input.bob$1
2 input //output$input.bob$2
3 input //output$input.bob$3
4 input //output$input.alice$0
5 input //output$input.alice$1
6 input //output$input.alice$2
7 input //output$input.alice$3
8 gate arity 2 table [1 0 0 0]
  inputs [4 5]
9 gate arity 2 table [0 1 1 0]
  inputs [4 5]
10 gate arity 2 table [0 1 0 0]
  inputs [8 6]
11 gate arity 2 table [1 0 0 1]
  inputs [8 6]
12 gate arity 2 table [1 0 0 1]
  inputs [10 7]
13 gate arity 2 table [0 0 0 1]
  inputs [4 0]
```

```
14 gate arity 3 table [0 0 0 1 0 1 1 1]
  inputs [13 9 1]
15 gate arity 3 table [0 0 0 1 0 1 1 1]
  inputs [14 11 2]
16 gate arity 2 table [0 1 1 0]
  inputs [12 3]
17 gate arity 2 table [0 1 1 0]
  inputs [15 16]
18 output gate arity 1 table [0 1]
  inputs [17] //output$output.alice$0
...
```

As we can see in the example every line is a wire, and every line has a number that corresponds to this wire. The declaration of a gate consists of a wirenumber to store the result, the arity and a table with the outputvalues of each permutation as in figure 8.2. The inputwires for a gate are written on the next line. The first eight lines are input lines, so the input can be two 4-bit integers. The output is now only one wire, representing a boolean, but Bob will have the same output. The comments are generated by the compiler. The precise input- and outputtypes are normally described in an extra file, like this:

```
Bob input integer "input.bob"
[0 1 2 3]
Alice input integer "input.alice"
[4 5 6 7]
Alice output integer "output.alice" [18]
Bob output integer "output.bob" [29]
```

8.2.3 Compiler

The compiler compiles SFDL into SHDL. Although it is an important part of the Fairplay project, it is also a bit beyond the scope of this chapter. So we will not look at it in details, but only look at the basic steps it uses.

1. **Parsing** First of all the lines are parsed into a usable memory format.
2. **Function inlining and loop unfolding** Functions are treated as macros and so every functioncall creates a copy of the function. Loops are unfolded, thats why is should have a fixed number of runs.
3. **Transformation into single-bit operations** Every mulit-bit operation is converted in a sequence of single-bit operations.
4. **Array acces handling** Arrays with compiletime constant indices are treated as normal variables. Arrays with expressions as index are more complex, but feasable using multiplexers.

5. **Single variable assignment** In boolean circuits each wire can be used only once, while in SFDL a variable can be assigned another value multiple times. To solve this the compiler transforms this into single assignments of more wires.

6. **Optimizations** A boolean circuit is allready derived, but it can be optimized. This includes peekhole optimization, duplicate code removal and dead code elimination.

8.3 2-party SFE

Now we have a language to describe functions, a compiler that can compile functions into single-pass boolean circuits in another language, we are able to look at the next important step, the protocol itself.

8.3.1 General working

Bob and Alice have agreed on a certain function, represented by a boolean circuit C. Because this circuit uses inputs from both Bob and Alice, they will have to coöperate to evalute C. This protocol uses one Oblivious Transfer (OT) for each inputwire.

To evaluate the function Bob creates M garbled circuits. How this is done is described in the next section. After that he sends these M circuits to Alice. Alice chooses at random one circuit to be evaluated. Bob now has to send the secrets of the other $M-1$ circuits. Alice then evaluates this $M-1$ circuits to see if they indeed represent the function f they want to evaluate. Using this cut-and-choose technique, malicious behaviour by Bob will be detected with a chance $1-1/M$. Now Alice can be sure that the circuit to be evaluated is correct, they start in a sequence of Oblivious Transfers, to find the inputs needed for the circuit. Alice then evaluates the circuit and sends Bob the results. A garbled circuit gives no information about the value of Bobs input, and nor does Oblivious Transfer.

8.3.2 Creating garbled circuits

For this section we use the notation $W_k, k = 0, ...l-1$ to represent all the wires of the boolean circuit C. Every gate has a single boolean output, and the number of inputwires can be 1, 2 or 3. The compiler doesn't create gates with a higher arity, so this is enough. In the explanation we will only look at binary gates, but unary and ternary gates work exactly the same.

The garbling of a circuit consists of the next two steps:

1. To create a garbled circuit, Bob has to make sure that Alice can't see the value of a wire. To ensure this Bob assigns to each wire $W_k \in C$ two random t-bit strings v_k^0, v_k^1. After that Bob assigns to each wire a bit p_k. This bit is appended to v_k^0 and v_k^1 as follows: $w_k^0 = v_k^0 \parallel (0 \oplus p_k)$ and $w_k^1 = v_k^1 \parallel (1 \oplus p_k)$.

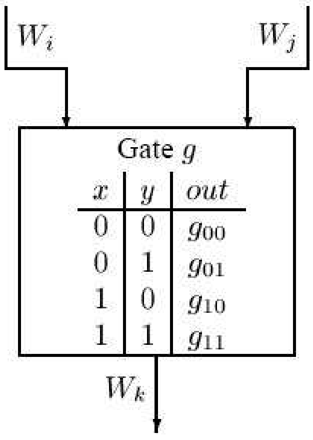


Figure 8.2: A GATE IN A BOOLEAN CIRCUIT

Instead of using 1 and 0 for wires, Bob uses the value w_k^0 for 0 and w_k^1 for 1. Alice can't find the real value of a wire, because w_k is randomly chosen. The length of bitstring v_k namely t is a security parameter, in the implementation set to 80.

2. Because the wires are garbled now, a gate also cannot see the correct values of an inputwire. So these gates should be changed so that they correspond to this garbled values. Also the gates may not reveal any information about the values that are used in it. So for each gate $g \in C$ whose inputwires are W_i and W_k and whose outputwire is W_j (figure 8.2) we perform the following steps:

   (a) The original truthtable of the gate consists of four entries. Bob creates a Garbled-Truth-Table (GTT) by replacing each 1 / 0 for the corresponding W_k^0 or W_k^1.

   (b) After that Bob encrypts the GTT into an Encrypted-Garbled-Truth-Table (EGTT). For entry (x, y) in g's GTT, define $x' = x \oplus p_i, y' = y \oplus p_j$. The entry is encrypted using v_i^x, v_j^y as encryption-keys and k, x', y' as an IV: $EGTT[x, y] = Encrypt_{v_i^x, v_j^y, k, x', y'}(GTT[x, y])$. The encryption is done by hashing $v_i^x \parallel k \parallel x' \parallel y'$ and $v_j^y \parallel k \parallel x' \parallel y'$ using SHA-1, and XORing the two results to the plaintext.

   (c) Bob constructs the Permuted-Encrypted-Garbled-Truth-Table (PEGGT) of g by swapping the entries in g's EGTT based ont het permutation bits assigned to g's input wires, namely p_i, p_j. Because Alice has a copy of the original truth table, values should stand in a different order. Alice could otherwise know the input values by comparing the two tables.

   (d) Now that the gates and wires are garbled, Alice doesn't know how to interpret her outputs anymore. So Bob sends for each outputwire of Alice an translationtable. In stead of just sending the values w^0 and w^1 Bob sends a one-way hash of these, so that Alice can verify their output without learning something about the values of the wires.

8.3.3 Evaluation garbled circuits

Evaluation of the garbled circuit is done in the following way. Bob sends his garbled inputs to Alice first. Alice can't learn anything from them, but she can fill them into the circuit. After that Alice gets her inputs by starting an Oblivious Transfer with Bob for each inputwire. Alice gets the garbled string she has to use, and Bob does not learn her values.

Now Alice evaluates the circuit gate by gate. For each gate Alice uses x, y as indices into an entry to be decrypted in g's PEGGT. Decryption is done by hashing $v_i \parallel k \parallel x \parallel y$ and $v_j \parallel k \parallel x \parallel y$ using SHA-1 and XORing the result with the ciphertext. w_k will be $Decrypt_{v_i,v_j,k,x,y}(PEGTT[x,y])$. During the evaluation Alice only sees garbled strings, so the system is safe. The outputs for Bob are sent to Bob, and Alice interprets her own outputs using the translation tables Bob sent her.

8.4 Security

The security of the system mainly depends on the security of the underlying principles:

- Pseudo-random function to create the garbled strings
- Oblivious Transfers
- SHA-1 hash function
- The fair termination of Alice

If the above parts function well, the whole system will function well. About the security we can get the following guarantees:

- Bob is guaranteed that an interaction with a malicious Alice is not different than an interaction with the trusted third party, except for a negligible error probability.
- Alice has the same guarantee with relation to Bob, with error probabilitiy of 1/m.

8.5 Applications

At the moment only one application of the Fairplay system is known. This application is Secure Computation of Surveys [RPFJ04]. It is an implementation of a system for conducting surveys while hiding the inputs of the respondents. It does not use the whole project, but only the second part. The compiler is not used because of the special requirements they had.

8.6 Summary

The Fairplay project implements 2-party Secure Function Evaluation in a general way. This might be a building block for many further applications. The biggest contribution of the project is the compiler that can convert functions into boolean circuits. This is because the protocol for 2-party SFE was allready known, and only the application took some time (i.e. 20 years).

Chapter 9

Proofs of partial knowledge

Written by *Thomas van Dijk.*

This chapter is about a paper by Cramer, Damgård and Schoenmaker: Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols [CDS94]. The paper is quite theoretical and operates on a meta level: given protocols and algorithms with certain properties, it composes them into a protocol with certain new properties. While this remains rather abstract, lets start with a possible application.

Say there is a company with n employees. The employees can send letters on behalf of the company, but only with at least k of them together. They are going to put a digital signature on the document with the following two properties.

- It will convince a verifier that indeed at least k employees participated.
- It will not reveal *which* of the employees took part. (E.g. because we want the company to be responsible for the message and make it impossible to it trace back to the individual employees.)

The first requirement would be easy: all k employees sign the document using some digital signature scheme. But this would reveal the identity of the signers. This chapter is about achieving the second requirement. (This application concerns identity protection, a goal which is covered more explicitly in Chapter 10 by Luis Faria.)

Being a little more specific, this is what we are going to see. There are n secrets and some subsets of those are 'qualified' —any k of them, in the previous example. Alice knows such a qualified set. Given zero knowledge proof protocols (with certain properties) for each of the secrets independently and a secret sharing scheme (with certain properties) for the so called "dual" of the qualified subsets, we will construct a new protocol that will allow Alice to convince a verifier that she knows *some* qualified subset of the secrets, without revealing which. The resulting protocol will itself not be zero knowledge, but it will have a property called 'witness hiding.'

First of all, we are going to see what 'witness hiding' means. This will also provide us with convenient vocabulary for the rest of the chapter. Then we will have a look at secret sharing schemes with emphasis on the properties we are going to need later on. Finally, we will see how to construct a proof of partial knowledge protocol.

9.1 Witness Hiding Protocols

Witness Hiding is a term introduced by Feige and Shamir in [FS90]. To explain what it is, we will need to know what a witness is in this context. First we'll define it in terms of a function.

Definition 9.1 (Witness) *Given a function f that is polynomial time computable, a value w is called a* witness *of x if and only if $x = f(w)$. That is, a witness is a pre-image.*

For example, if we take f to be exponentiation in $\mathbb{Z}_p$ for prime p, a witness of x is its discrete log. If f is clear from the context, it is often not mentioned. In this example every value has only one witness, because it only has one discrete log. But if f is not injective, values will have multiple witnesses. With this observation, we define the concept of a witness set.

Definition 9.2 (Witness set) *Given a function f that is polynomial time computable, the witness set $w(x) = \{\ w \mid f(w) = x\ \}$. That is, the witness set for a value consists of all its witnesses.*

The requirement that f is polynomial time computable says that given a value and a supposed witness, it can be efficiently checked whether it really is. Though there is no claim about the difficulty of finding witnesses given an x, we are interested in functions for which this is hard. That, after all, is what big parts of cryptography are based on: one-way functions. An alternative wording of the definition of witness –which is actually more like the one in most papers– closer reflects this.

Definition 9.3 (Witness—in terms of a relation) *Given a relation $R = \{\ (x, w)\ \}$, for which membership can be tested in polynomial time, w is a witness of x if and only if $(x, w) \in R$.*

A lot of cryptographic protocols use the concept of public and private information. Encryption using public key cryptography, for example, is well known. The message is encrypted using the public key of the intended recipient and we can trust that only someone who knows the corresponding private key can read the message. In our newly acquired vocabulary: we can trust that only someone who knows a witness of the public key can read the message. So the relation in the above definition can be seen as the set of valid public key, private key pairs.

9.1.1 Proofs of knowledge

Zero knowledge proofs are a very interesting concept in cryptography. It is usually a conversation between two parties, the prover P and the verifier V, in which P wants to convince V that she knows a certain secret. Just telling the secret would be very convincing, but this is usually not an option as it would typically defeat our cryptographic goals. This is where zero knowledge proofs come in.

Casus 9.1: IDENTIFICATION USING SCHNORR'S PROTOCOL

Schnorr's protocol can be used for identification. Alice has a public value that represents their identity and the protocol consists of her proving she knows the discrete log of that value. That is, she claims to know a witness of her public key. This setup is safe under the assumption that calculating a discrete log is intractable, which is generally accepted (though reluctantly, because it remains only an assumption). People cannot calculate a witness from Alice's public value, but she herself can construct her public value in such a way that she knows a witness. This is what links 'knowing a witness' to 'being Alice.' A short summary of the protocol follows.
All calculations are in $\mathbb{Z}_p$ and let g be an element of order q, where p and q are large primes.

- Alice contacts Bob and claims that she is indeed Alice. Bob looks up Alice's public value x.
- Alice chooses a random r and sends $s = g^r$.
- Bob sends a random c, the challenge.
- Alice replies with $y = r + w \cdot c$, where w is her witness of x (that is, $\log x$).
- Bob accepts iff $g^y = s \cdot x^c$.

Take special notice that this conversation is convincing to Bob because he knows he has sent his challenge only after Alice committed herself to a value of s.

First of all, P and V both know a value x and P wants to convince V that she knows a witness of x. This is what makes it a proof of knowledge. Secondly, nobody is allowed to learn *anything at all* from observing the conversation or even cheating the protocol. This is what makes it zero knowledge. A zero knowledge proof (ZKP) has both these properties.

We will now look at several relaxations of the second requirement. One that we will be particularly interested in later on is called honest verifier zero knowledge (HVZK). For this it is only required that a verifier that follows the protocol, i.e. does not cheat, does not learn anything from his conversation with P. (Note that, unlike ZK, this does not require anything regarding third parties.) This is a weaker property and protocols with it are typically much easier to design than fully ZK protocols.

Now we get back to [FS90]. We still consider protocols to prove knowledge of a witness and assume that the public value under consideration has multiple witnesses. If, from observing the conversation or even cheating the protocol, nobody can learn *which* of the witnesses was used by the prover, the protocol is called witness indistinguishable (WI). In itself this is not a very interesting property (it doesn't require secrecy about the value of the witnesses!), but it is a nice building block for several theorems and schemes.

If, from observing the conversation or even cheating the protocol, nobody can learn the value of any witnesses they did not know before the protocol started, the protocol is called witness hiding (WH). This is a much more interesting property and at first sight one might even confuse it with zero knowledge. But ZK prohibits *any information at all* from leaking, whereas WH only prohibits information about the witnesses from leaking.

9.1.2 Zero knowledge versus Witness hiding

The reason these alternative notions have been introduced is because there is a serious drawback to the zero knowledge requirement, especially so in our goal of constructing a proof of partial knowledge protocol. Observe the following theorem from [FS90].

Theorem 9.4 *There exists a zero knowledge proof protocol $\mathcal{P}$ for the discrete log problem, which when executed twice in parallel discloses the discrete log of the input.*

They prove this by constructing a protocol which is zero knowledge, but when run twice in parallel allows a cheating verifier to learn the witness very easily. While their protocol is rather contrived, it does show that the zero knowledge property in general is not preserved under multiple parallel executions. This means that when using multiple zero knowledge proofs as building blocks in a cryptographic scheme, one has to be very careful that the proofs are still zero knowledge in this composition. As a result, such protocols can hardly ever be reused and always have to be carefully designed up to the smallest detail for the specific task at hand.

Luckily there are interesting relations between the concepts introduced above and these will help us in our goal of constructing a proof of partial knowledge protocol. For the proofs of these, as well as formal definitions of the concepts, refer to the papers in question.

Proposition 9.5 *[CDS94] Let $\mathcal{P}$ be a three round public coin proof of knowledge for relation R. If $\mathcal{P}$ is honest verifier zero knowledge, then $\mathcal{P}$ is witness indistinguishable.*

Two clarification are called for. “Public coin” here means that the only thing the verifier will ever say is random numbers. “Three round” means consisting of three rounds of communication (though the result is generalizable to more rounds). So for example, Schnorr’s discrete log protocol fits this proposition:

- it is honest verifier zero knowledge.
- it is “public coin:” V only sends a random number.
- it consists of three rounds: P sends s, V sends the challenge, P responds.

Therefore. by proposition 9.5, Schnorr’s protocol is witness indistinguishable.

The previous proposition related honest verifier zero knowledge to witness indistinguishable. Now we will relate the latter to witness hiding.

Proposition 9.6 *[FS90] If $\mathcal{P}$ is witness indistinguishable and $w(x)$ contains at least two independent witnesses, then $\mathcal{P}$ is witness hiding.*

This proposition shows us that WI is no dead end; showing a protocol to be WI can get us somewhere. And now, to make WI a truly useful concept, we have the following.

Theorem 9.7 *[FS90] WI is preserved under polynomial composition of protocols.*

Polynomial composition here means any composition you would like, including arbitrary interleavings of steps, but with not more protocol instances than polynomially many in the size of the secret. For our purpose, this is not a restriction in practice.

We are now ready to sketch the first half of our proof of partial knowledge protocol. There are n publicly known values and there is a public-coin protocol $\mathcal{P}$ that can be used to prove knowledge of corresponding witnesses and that is *honest verifier zero knowledge*. Alice knows a witness for some of the values and for the others she doesn't. Now comes a surprising point: she is going to give a proof of knowledge of all of them. For the secrets she knows, she can use $\mathcal{P}$ to give one, but what about the others? She doesn't actually know the secret, so how is she going to prove she knows it? She is not actually supposed to be able to do that! Indeed, she is not going to play $\mathcal{P}$ the usual way —first commit herself, then receive a challenge, then reply. Instead, she will use $\mathcal{P}$'s *simulator* (it has to have one, otherwise it would not be zero knowledge): a procedure for efficiently constructing 'fake proofs,' that works by being able to freely choose the challenge instead of being dictated a challenge only after the first message has been fixed.

Alice is able to give n transcripts of $\mathcal{P}$, one for each public value. Either by really using $\mathcal{P}$ and her corresponding witness, or by using $\mathcal{P}$'s simulator. These form n parallel executions of $\mathcal{P}$, and $\mathcal{P}$ is (honest verifier) zero knowledge so, because of theorem 9.4, we should be wary of this composition. But now the work we have done so far is going to pay off.

1. Since $\mathcal{P}$ is HVZK and public-coin, it is also WI.

2. WI protocols remain WI in compositions, so each of the n instances of $\mathcal{P}$ will still be WI when used together.

3. All of those instances whose corresponding public value has at least two independent witnesses are WH. (If this isn't the case already, [FS90] provides a technique to transform the protocol and the involved values so this *is* the case.)

So we end up doing n parallel executions of $\mathcal{P}$ and those are all witness hiding, so nobody learns anything about any witnesses.

We have a big problem still: right now Alice could use the simulator every time and play the whole protocol through without knowing any witnesses. Using techniques from secret sharing, we will now sketch a way for Alice to prove that some of the transcripts weren't produced by the simulator.

9.2 Secret Sharing Schemes

This will not be a proper treatment of secret sharing schemes by any account. Instead we just need some vocabulary and some results and are going straight for those.

A secret sharing scheme divides a secret s into n shares and some subsets of the shares can be used to reconstruct the secret. A subset that can reconstruct the secret is called a *qualified* set.

A well known example of a secret sharing scheme is the additive scheme. Here the shares are chosen in such a way that their sum equals the secret. Only if all the shares are put together will the value of the secret be known. Which means that only the full set of shares is qualified.

Another well known secret sharing scheme is Shamir's polynomial scheme [Sha79]. Here all n shares lie on a polynomial of degree $k - 1$ and the secret is defined as the value of this polynomial at $x = 0$. This way, any k of the shares uniquely define the polynomial, which allows the secret to be computed. Here, all sets of k shares are qualified.

The *access structure* of a secret sharing scheme is the collection of all its qualified sets. The access structure for the additive scheme is a singleton containing the set of all shares; for the polynomial scheme it is the collection of all subsets of at least k shares.

Access structures can also be considered the other way around: not as a property of a scheme, but as a specification for a whole family of schemes. In this view, the polynomial scheme is just one of probably many schemes implementing the 'k-out-of-n' access structure. From now on, we are not going to care about specific schemes, but only about access structures; we trust appropriate schemes will exist. (In fact, techniques are known to construct explicit schemes from almost any reasonable access structure —and even quite some unreasonable ones.)

One can define the *dual* of an access structure and while the details of this are vitally important to the actual working of the final protocol, they are not particularly interesting for our current exposition.

Finally, there are two restrictions on secret sharing schemes that we will use.

- Given a secret and all the shares, it should be possible to efficiently check whether the shares are consistent with the secret. (In the additive scheme, whether the shares add up to the secret; in the polynomial scheme whether all shares lie on the same polynomial with the proper degree the right intersection with the y-axis.)

- Given a secret and a non-qualified set of shares, it is possible to efficiently 'make up' more shares so that the existing shares and those new shares together reconstruct to the secret.

9.3 Proofs of Partial Knowledge.

Now we are ready to build our proof of partial knowledge protocol. There are n public values and we are given a corresponding three-round public coin HVZK protocol $\mathcal{P}$ for secrets of that kind. We are also given an access structure. Our protocol is going to allow Alice to prove she knows a set of witnesses that is a qualified set according to that access structure. Let A be the set for which she knows a witness and $\bar{A}$ those for which she doesn't. The algorithm will now be as follows.

Round 1 For all the secrets in A, Alice knows a witness. So for those, she can use $\mathcal{P}$ to determine her first message to send.
For the others, she can't do that: she doesn't know a witness so she is going to be in trouble if Bob sends her a challenge. For those, she runs the simulator of $\mathcal{P}$ to get a triple (m_1, c, m_2).
She now has the first message of a ZKP for every secret. She sends them all to Bob.

Round 2 Bob sends back a (suitably big) random number s. This is his challenge, only we don't call it c to avoid confusion with the challenges in $\mathcal{P}$. (Notice that our new protocol is public-coin as well.)

Round 3 For the secrets in $\bar{A}$, the challenge for $\mathcal{P}$ has already been decided by the simulator and the third message is already known as well.
This leaves the challenges for the proofs concerning A. For those proofs to actually mean anything, their challenges have to depend on Bob's challenge s. This is where the secret sharing comes in. We are going to interpret the challenges that occur in the simulator's proofs as shares of Bob's challenge s. Without further elaboration here, these shares form a non-qualified set in the dual of the access structure that our protocol must exhibit. By a requirement on secret sharing schemes, given a secret and a non-qualified set of shares, it is possible to efficiently 'make up' more shares so that the existing shares and those new shares together reconstruct to the secret. Alice does this now using s as the secret and *the simulator's* challenges as non-qualifying set of shares. This way she creates a share for each of the proofs about a secret in A and she uses these shares as the challenges for $\mathcal{P}$. She actually knows a witness of the corresponding public value, so she can finish $\mathcal{P}$ properly.
She now has a challenge for all her ZKPs (either from the simulator or from constructing more shares to fit Bob's challenge s) and she also has $\mathcal{P}$'s response for them all (either from the simulator as well or because she knew a proper witness). She sends all these to Bob.

Bob now verifies two things. First of all, he checks that all the ZKPs are correct (that is, would be accepted by $\mathcal{P}$'s verifier). But this in itself has no convincing power, as has been mentioned at the end of first section of this chapter: they might all come from a simulator. The convincing power of a ZKP comes from the fact that the prover sends his first message before he knows the challenge. So Bob checks that all the n

challenges together reconstruct to his challenge s (which can be done efficiently because of a restriction on secret sharing schemes). This convinces him that the proofs that did not come from a simulator must form a qualified set: otherwise the challenges from the simulator's proofs would have formed a qualified set in the *dual* of the access structure, and Alice wouldn't have been able to make all the challenges together reconstruct to his challenge in Round 3.

If both checks pass, he accepts Alice knows witnesses to a qualified set of secrets.

With $\mathcal{P}$ and the public values according to some digital signature scheme, this protocol allows the construction of a system with the requirements set out in the introduction of this chapter: in a company with n employees, any k of them together can sign a letter in such a way that it will convince a verifier that indeed at least k employees cooperated in signing, but that will not reveal the identity of the individual employees.

Chapter 10

The Sigma Protocol

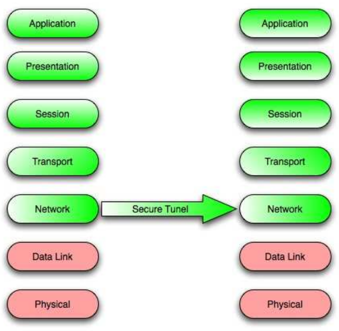
Written by *Luis Faria.*

Although everyone takes privacy in normal life for granted, trying to get the same level of privacy on the Internet (or even on your own computer) is a little less accepted, and sometimes a bit more complicated. In this chapter I will talk about one example of protocol that provide identity protection while exchanging keys in the internet.

Other protocol that shows a different situation in which identity protection is provided is referred in Chapter 9. The two protocols are very different, and they show different points of view of identity protection, complementing each other on a larger view.

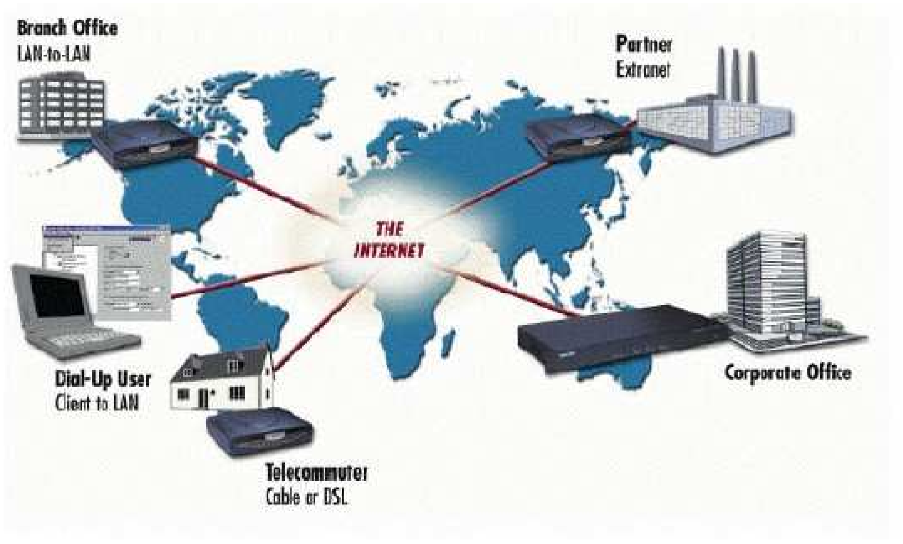
Before I talk about the Sigma Protocol let's contextualize it. First I talk about IPsec suite of security protocols, then the Internet Key Exchange (IKE) protocol used in IPsec, and finally the sigma protocol that is the base for the IKE signature mode.

10.1 Context of Use



IPsec is a standard for securing Internet Protocol (IP) communications encrypting and/or authenticating all IP packets. It replaces the IP protocol on the protocol stack, ensuring security to all above layers. This protocol suite is a set of cryptographic protocols that provide secure communication (secure packet flow, as the packet is the basic unit of the Network layer) between two points. This two points (or parties) are related by a Security Association (SA), that is, a set of parameters, algorithms and shared keys agreed by the two parties and locally stored by them.

IPsec would be nearly useless without the cryptographic facilities of authentication and encryption, and these require the use of secret keys known to the participants but not anyone else. The most obvious and straightforward way to establish these secrets is via manual configuration: one party generates a set of secrets, and conveys them to all the partners. All parties install these secrets in their appropriate Security Associations (SAs).



But IPsec is oriented to Wide Area Networks, such as internet, so the Manual Configuration does not scale well, nor it is secure. Conveying the secrets to all the partners, which can be on the other side of the world, is not practical, therefore a protocol is needed to allow sharing the secrets to the partners by the internet. As internet is not a secure way of transmitting secrets, care has to be taken to don't allow other parties to see the secret, or even to give the secret to the wrong party. So in this protocol we need cryptographic tools to prevent others from knowing our secret and to identify the party we are giving the secret to. The only key exchange protocol defined for IPsec is the Internet Key Exchange (IKE).

The IKE protocol exists to allow two endpoints to properly set up their Security Associations, including the secrets to be used. There are some cryptographic properties that we want this protocol to have:

- **Authenticated Key Exchange**
  Each party of a IKE session needs to be able to uniquely verify the identity of the peer with which the session key is exchanged.
- **Perfect Forward Secrecy**
  The compromise of a session key or long-term private key after a given session does not cause the compromise of any earlier sessions.
- **Identity Protection**
  Hiding parties identities from passive or active attacker in the network

This last property, Identity Protection, is the one that is more interesting, as it is the biggest advantage of Sigma Protocol against the former used Key Exchange protocols.

Consider a passive attacker an eavesdropper, that has access to the messages transmitted between the parties, but cannot create, modify or delete any of the messages transmitted. A active attacker is a *"man-in-the-middle"* with full control of the communication links between parties. Identity Protection is referred in Section 10.2.

IKE was made for general purpose, so there are different modes allowed, with different characteristics, that fit different needs of protection and identity protection. IKE provides several key-exchange mechanisms:

- Long term pre-shared symmetric key
- Public-key encryption
- Digital Signature
- Re-key

We are interested in the mechanism that provides identity protection, the Digital Signature mechanism. IKE signature modes are particular cases of a larger family of protocols called Sigma, which is the subject of this Chapter.

10.2 Identity Protection

As referred in Section 10.1, care has to be taken so the secret is not seen or transmitted to the wrong party. Therefore a Key Exchange protocol needs strong mutual authentication, and consequently the identity of each party has to be proved (and transmitted) to the other, through the internet. But in some cases, disclosure of the identities over the network may be unwanted.

Attackers over the network can use *Identity Probing attacks* to discover the identities of the responders. The Sigma Protocol offers protection against this kind of attacks, protecting responders from passive an active attacks. But in some cases the interest can be the protection of the initiator. Sigma Protocol can also offer protection of the initiator from passive and active attacks. But it is not possible to design a protocol that will protect both parties against active attacks, because the first party to authenticate himself needs to disclosure its identity to the other before he can verify the identity of the latter. Sigma offers the best protection possible, in two variants:

- Σ_I **variant** (Section 10.5)
  Protects the *initiator* from passive and active attackers and the *responder* only from passive.
- Σ_R **variant**(Section 10.6)
  Protects the *responder* from passive and active attackers and the *initiator* only from passive.

On the other hand Sigma protocol is also suitable when no identity protection is needed, and that is the core of Sigma Protocol (Σ_0), which will be explained in Section 10.4

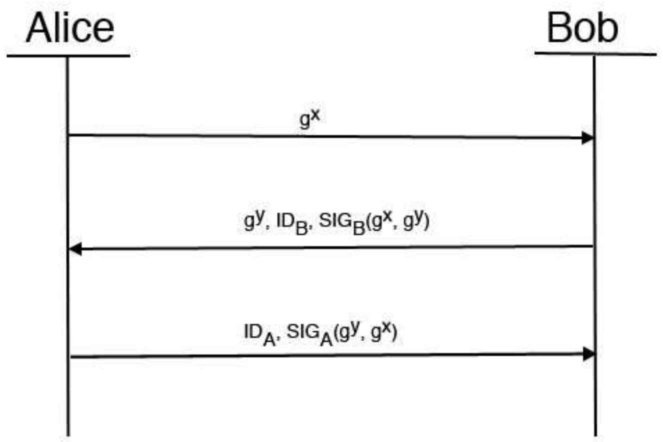
Another aspect to notice on identity protection is traceability. When making a successful session of the key exchange protocol, the receiver party can get a undeniable proof that a session has occurred. This can be an unwanted feature for the initiator. Sigma provides good solution for it, has it doesn't need to sign the receiver identity on the response message, as it occurs in ISO protocol (see Section 10.3.3).

10.3 Protocols Sigma was based on

The Sigma protocol is based in former used protocols, gathering their strengths and learning by their weaknesses. Next there will be a overview of the aspects that Sigma took from these protocols, presenting an incremental rational to the Sigma base protocol itself.

10.3.1 Identity-misbinding attack

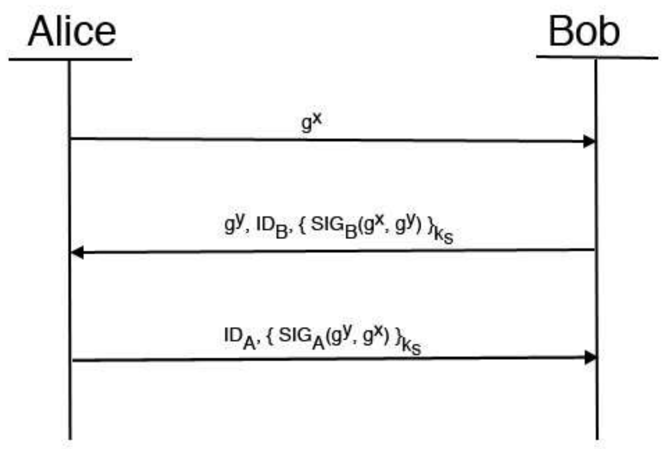
In [Kra03] it is presented an introductory protocol designated by BADH ("badly authenticated DH") which is the most intuitive solution for authenticated key exchange:



This is the most basic authenticated DH exchange using digital signatures. Denote ID_A and ID_B as the certificated identities of Alice and Bob, respectively. The inclusion of g^x under SIG_B and g^y under SIG_A is crucial to protect from replay attacks, but this protocol is defenseless against identity-misbinding attacks. An active attacker called Oscar, could let the first two messages untouched and replace the third message with: $ID_O, SIG_O(g^y, g^x)$. As a result Alice thinks that she is talking with Bob, but Bob thinks he's talking with Oscar.

10.3.2 STS protocols

A protocol that tries to solve the identity-misbinding attack is the basic STS protocol. This protocol uses encryption to try to solve the problem:

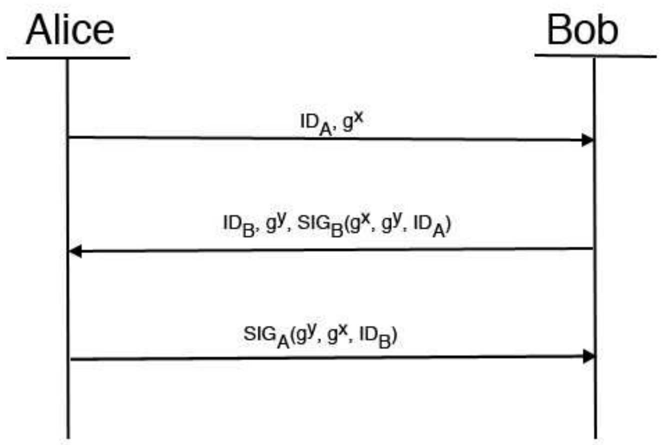

Alice
Bob
g^x
g^y, ID_B, { $SIG_B(g^x, g^y)$ }$_{K_S}$
ID_A, { $SIG_A(g^y, g^x)$ }$_{K_S}$

In STS the signature is encrypted using the key output of the session (K_s). Unfortunately this is not enough to bind the identities with a result key, failing to defend from identity-misbinding attacks when the Certificate issuer does not provide *proof of possession*. Oscar can register Alice's public key as it own and replace ID_A by ID_O in the finish message.

On ISO and Sigma protocol there is no need for external proof of possession.

10.3.3 ISO protocol

The ISO protocol solves the identity-misbinding problem adding the identity of the intended recipient of the signature to the message that is signed.



This is a secure protocol, but it is not suited for identity protection cause the protocol itself instructs to sign the recipients identity. Therefore none of the parties can authenticate himself before knowing the other parties identity. This is the greatest advantage of Sigma.

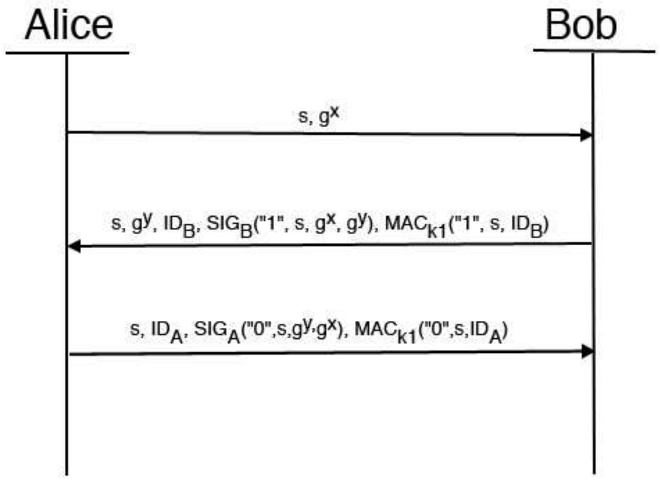
10.4 The core Sigma Protocol: Σ_0

The ISO protocol is not suitable for identity protection (Section 10.3.3), because it uses the signing of the peer identity to bind the key and the session-id. But Sigma protocols have a very different approach to authentication. Instead of the signing of the other parties identity, a MAC-based mechanism is added to "compensate" for the unsigned parties identity. A Message Authentication Code is actually better suited for this purpose. A pseudo-ramdom function (PRF), based on the resulting DH-exponential g^{xy}, is used to generate keys (k_0, k_1, k_2), ensuring that they are independent from each other (PRF is a one-way function).

First it will be presented a basic version of Sigma, introducing the core of the protocol. Next, the considerations about identity protection will be taken into account in the form of variants of this core protocol.

Let's consider some initial information:

- p, q are primes, $q/p-1$, and g has order q in $\mathbb{Z}_p^*$
- Each player has a private key for SIG
- All players have the public verification keys of the other players
- All players have access to a Message Authentication Code function (MAC)
- All players have access to a Pseudo-random function (PRF)


Alice
Bob
s, g^x
s, g^y, ID_B, SIG_B("1", s, g^x, g^y), MAC_k1("1", s, ID_B)
s, ID_A, SIG_A("0",s,g^y,g^x), MAC_k1("0",s,ID_A)

In the presentation of this protocol more details are shown than in the former ones. It's explicitly shown the session-id, denoted by s, that uniquely identifies this session, and it is checked for uniqueness by both parties. Next it is explained each message, being the first one, from Alice to Bob, denoted by *Start Message*, the second, from Bob to Alice, *Response Message*, and the last one, from Alice back to Bob, *Finish Message*.

1. **Start Message**
   Alice computes a session-id s that was never used in previous successful or unsuccessful sessions. The DH-exponent g^x is computed with $x \overset{R}{\longleftarrow} \mathbb{Z}_q^*$. Is stored in the state of session (ID_A, S).

2. **Response Message**
   Bob checks if s is unique, then it computes $g^y, y \overset{R}{\longleftarrow} \mathbb{Z}_q^*$ and computes g^{xy} by calculating the power of the received exponential and his secret y, because $g^{xy} = (g^x)^y$. Next he computes the signature and the MAC_{k_1} value, where $k_1 = PRF_{g^{xy}}(1)$. The secret will be $k_0 = PRF_{g^{xy}}(0)$. As a security measure, Bob has the care to safely erase y and g^{xy} from local storage. Finally Bob sends the response message.

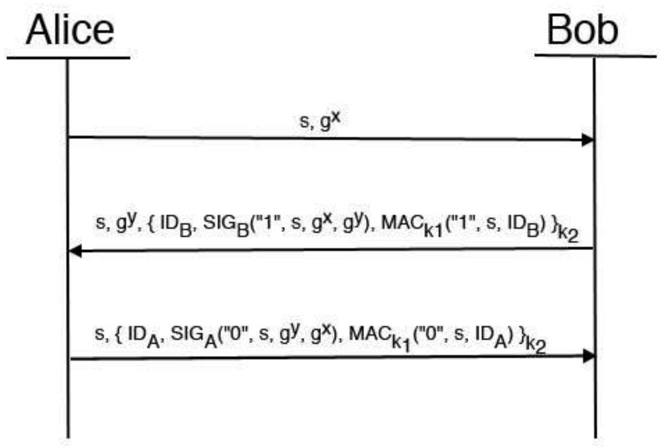
3. **Finish Message**
   Alice retrieves Bob public key and verifies Bob's identity, knowing that she is talking to him. She also checks MAC_{k_1}, binding Bob to the session-id and the secret. Now she creates the finish message, by computing the signature and the MAC_{k_1} value. Finally she computes and keeps the secret k_0, completes the session with public output (ID_A, s, ID_B), erases the session state and sends the finish message.

   By Bob's turn, he verifies the signature and the MAC_{k_1} value, if it passes all checks erases the session with the public output (ID_B, s, ID_A).

The strings "0" and "1" are tags, and intent to separate authentication information created by the initiator and the responder. They serve as *symmetry breakers*, to prevent reflection attacks in some of the variants of the protocol.

10.5 Identity Protection variant: Σ_I

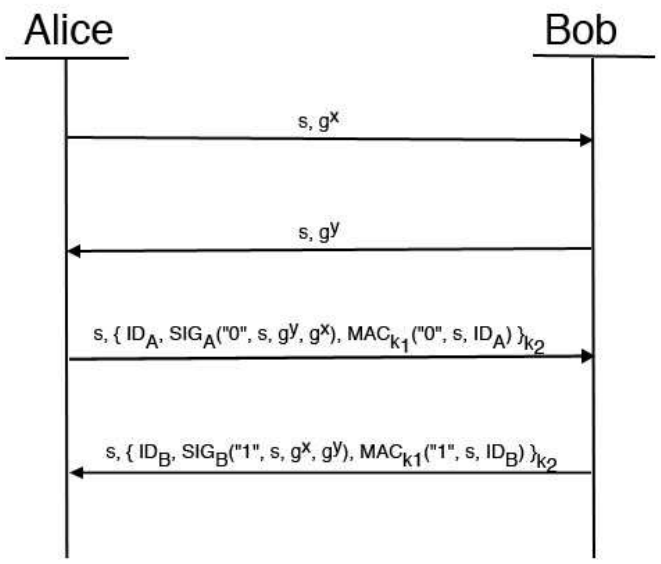
It is obvious that Σ_0 does not provide identity protection, because Bob sends his identity on the response message. But it can be easily added by simply encrypting identities and signatures using a key $k_2 = PRF_{g^{xy}}(2)$.


Alice
Bob
s, g^x
s, g^y, { ID_B, SIG_B("1", s, g^x, g^y), MAC_{k1}("1", s, ID_B) $\}_{k2}$
s, { ID_A, SIG_A("0", s, g^y, g^x), MAC_{k1}("0", s, ID_A) $\}_{k2}$

This protocol is safe against passive attacker, as no information can be taken from the messages, has they are encrypted. An eavesdropper will never know which identities participated in the protocol. But only Alice is safe against active attacker, because Bob has to authenticate himself to Alice in the response message before he knows Alice identity. But in some applications, protecting the Responders identity can be more important. The Σ_R variant considers that.

10.6 Four Message variant: Σ_R

Bob can delay the sending and authentication information to a fourth message, after he verifies Alice's identity. In this way he will be protected against active attackers. Alice looses the protection against active attackers, but still has protection against passive ones.


Alice
Bob
s, g^x
s, g^y
s, { ID_A, SIG_A("0", s, g^y, g^x), MAC_k1("0", s, ID_A) }_k2
s, { ID_B, SIG_B("1", s, g^x, g^y), MAC_k1("1", s, ID_B) }_k2

Now we can see the use of the tags, giving a sense of direction to the communication, and preventing this protocol to be open to a simple replay attack, returning the messages back to Alice, and making her accept a session with herself.

10.7 Zero Knowledge Proof of Σ_0

A Zero knowledge proof is an interactive method for one party to prove to another that a statement is true, without revealing anything other than the veracity of the statement. In this section I will give some considerations about the zero knowledge proof properties of Σ_0.

Zero Knowledge has three properties:

- **Acceptance**
  If Alice has the secret and is honest with the protocol than she will be accepted by Bob, and vice-versa. In Σ_0 each party identity will be proven to each other (i.e. they have their private key).

- **Rejectance**
  If a party is accepted by Bob with a probability greater then κ (probability of successful cheating), than this party has the secret. This will be κ is a security parameter.

- **Zero-Knowledge**
  Any party can simulate the protocol (knowing the public keys)

The only aspect of the protocol that can be viewed as a zero-knowledge proof is the authentication. The result of the Diffie-Helman exchange is part of the protocol proof realm, and has nothing to do with Zero Knowledge Proof. So, the only part of the messages transmitted that have interest to this proof are the signatures. The most important feature that has to be observed is the uniqueness of session-id s. This feature makes the message that is signed also unique. This is very important to assure that a signature will not be done twice (and Oscar could reutilize it).

For acceptance and rejectance, we can transport the proof to the signature itself, that surely will provide a proof for them. If both parties are honest with the signing scheme, they will be accepted by each other. If Oscar as a probability greater than κ, being κ a security parameter of the security scheme, then he has surely the secret.

For zero knowledge, we should try to find a simulator for the signature scheme. Oscar can choose random $\mathcal{S}$, compute $\mathcal{M}$ such that $SIG_A(\mathcal{M}) = \mathcal{S}$ and then from $\mathcal{M}$ extract s, g^x and g^y. But the possibility that this message cannot be distinguished of a real one is a discussable subject. For example, we now that a message has a tag, with two alternatives $\{0,1\}$. Assuming the representation has 1 bit, we have 50% of probability to get it right. But that assumption can be wrong. Some kind of pattern could be needed if "0" and "1" had more complex representations. But a possibility for zero-knowledge and consequently zero knowledge proof can be taken from here.

Chapter 11

E-voting

Written by *Maarten Kous and Michel Sluis.*

We have all witnessed the disaster that was the American election in 2000. These elections showed us that voting with paper ballots can give undesired results, and it resulted in a desire for technology which makes it possible for for every voter to enable her to verify that the vote she cast was counted correctly, and also prohibit the risk of fraud or ambiguous ballots. In this paper we will discuss the requirements that electronic voting schemes must have to make it possible to vote electronically, this could be via a voting machine or even over the internet. Then we will discuss a few electronic voting schemes who try to contain as much of the discussed requirements as possible.

11.1 Requirements on voting schemes

Elections are a very important and difficult subject. This is of course because of the major implication the results can have. Therefore, in order to successfully design an electronic voting scheme, a lot of requirements have to be met. Below, we will explain the requirements that are set for such schemes.

1. **Completeness** All votes are counted correctly.
2. **Soundness** A dishonest voter cannot disrupt the voting process.
3. **Privacy** All votes must be kept secret, i.e., no one can discover how a voter voted.
4. **Receipt-Freeness** This is a stronger requirement of privacy. Not only must all votes be kept secret, a voter must not be able to prove how she voted. This could lead to vote buying, or coercion.
5. **Unreusability** No voter must be able to vote twice.
6. **Eligibility** All voters who are allowed to vote, can vote.

7. **Fairness** Nothing must affect the voting process.
8. **Verifiability** The results of the election can be verified. This can de done by the voter herself (individual verifiability) and by all other parties (universal verifiability).
9. **Robustness** A voting scheme must allow for a certain amount of faulty players. This includes all concerning parties in the protocol, such as voters, registrars, talliers, etc.

11.2 Example of a blind signature voting scheme

In this section we will show an example of a blind signature voting scheme proposed by Fujioka et. al [FOO93]. It is a very simple scheme that doesnt require a lot of communication between the different parties involved, namely the voters, an administrator and a counting center. Apart from a blind signature scheme also an ordinary signature and a bit commitment scheme are used in the voting process. The scheme takes into account security proporties like the privacy of the voter, the soundness of the scheme and verifiability of the voting process.

11.2.1 Notations

The scheme uses the following notations:
V_i: Voter i
A: Administrator
C: Counter
$\xi(v, k)$: Bit Commitment scheme for message v using key k
$\sigma_i(M)$: Voter V_i's signature scheme
$\sigma_A(M)$: Administrator's signature scheme
$\chi_A(m, r)$: Blinding technique for message m and random number r
$\delta_A(s, r)$: Retrieving technique of blind signature
ID_i: Voter V_i's identication
v_i: Vote of voter V_i

11.2.2 Voting model

This section will explain the scheme in detail, going through the various stages of the voting process. In short the scheme works like this: A voter prepares his ballot, gets approval from the administrator so that he can vote and send in his ballot to the counter via an anonymous channel. The administrator only gives his key to a voter if the voter is registered, allowed to vote and hasn't voted before. The counter only collects votes, and counts them when all voters have cast their vote. The actual vote is at all times only known to the voter, and cannot be seen by other parties until after vote counting because of the bit-commitment scheme. A bit commitment scheme is a way of encrypting a message for later verification by another party. An example of a bit commitment scheme is the following:

If b is the message to be committed, Alice generates a large random number r and gives Bob the hash of b concatenated with r. To open her commitment, Alice reveals b and r thus letting Bob recalculate the hash and compare it with the hash given him earlier to make sure Alice didn't cheat.

The actual voting process is as follows:

- Preparation
  - Voter V_i selects vote v_i and completes the ballot $x_i = \xi(v_i, k_i)$ using a key k_i randomly chosen.
  - V_i computes the message e_i using blinding technique $e_i = \chi_A(x_i, r_i)$.
  - V_i signs $s_i = \sigma(e_i)$ to e_i and sends (ID_i, e_i, s_i) to the administrator.
- Administration
  - Administrator A checks that the voter V_i has the right to vote. If V_i doesn't have the right, A rejects the administration.
  - A checks that V_i hasn't voted before. If V_i has already applied for a signature A rejects the administration.
  - A checks the signature s_i of message e_i. If they are valid, then A signs $d_i = \delta_A(e_i)$ to e_i and sends d_i as A's certificate to V_i.
  - At the end of the administration stage A announces the number of voters who were given the administrator's signature, and publishes a list that contains $\{(ID_i, e_i, s_i)\}$.
- Voting
  - Voter V_i retrieves the desired signature y_i of the ballot x_i by $y_i = \delta(d_i, r_i)$.
  - V_i checks that y_i is the administrator's signature of x_i. If the check fails V_i claims it by showing that (x_i, y_i) is invalid.
  - V_i sends (x_i, y_i) to the counter through an anonymous communication channel. This channel prevents the counter from linking the vote to the voter, thus ensuring voter privacy.
- Collecting
  - Counter C checks the signature y_i of the ballot x_i using the administrator's verification key. If the check succeeds, C enters (l, x_i, y_i) into a list where l is a number that voter V_i can later link his key k_i to during the opening stage.
  - After all voters have voted C publishes the list of votes, which is public to all parties involved.

- Opening
  - Voter V_i checks that the number of voters on the counter's list is equal to the total number of voters. If the check fails, voters claim this by opening r_i used in encryption.
  - V_i checks that his ballot is listed on the vote list. If his vote is not listed V_i claims this by opening (x_i, y_i), the valid ballot and its signature.
  - V_i sends his number and key (l, k_i) to the counter through the anonymous communication channel.
- Counting
  - Counter C opens the commitment of the ballot x_i and retrieves the vote v_i (the voter doesn't have to send his actual vote, C can check which candidate matches the hash code in x_i using k_i).
  - C counts the voting and announces the voting results.

11.2.3 Proof of security

The authors claim their scheme is secure because it has the following properties:

- Completeness: All valid votes are counted correctly.
- Soundness: The dishonest voter cannot disrupt the voting.
- Privacy: All votes must be secret.
- Unreusability: No voter can vote twice.
- Eligibility: No one who isn't allowed to vote can vote twice.
- Fairness: Nothing must affect the voting.
- Verifiability: No one can falsify the result of the voting.

The author goes on to prove all of these properties, but here we shall show the most important ones.

Theorem 11.1 *Soundness: Even if a voter intends to disrupt the election there is no way to do it.*

Proof. The only way to disrupt the election is for the voter to keep sending invalid ballots. However, This can be detected in the Counting stage. It can be assumed that a voter sends an illegal key, so his vote cannot be opened. In this situation we cannot distinguish between a dishonest voter or counter. To solve this problem a voter should send his key to several independant parties (for example the candidates of the election), who are assumed not to collaborate. $\triangle$

Theorem 11.2 *Verifiability: Assume that there is no voter who abstains from voting and no one can forge the ordinary digital signature scheme. Then, even if the administrator and counter conspire, they cannot change the result of the voting.*

Proof. It is clear that the only disruption is for the counter not to add a valid ballot to the list. However, this can easily be pointed out by the voter whose valid ballot isn't contained in the list. So we will only consider disruption by the administrator in the following.

If there is no voter who abstains from voting (he sends his ballot even if he abstains), there is no way for the administrator to dummy vote. So only valid voters can vote and the result is trustable.

If the list overflows, every voter claims that he is an iligible voter and he was given a valid signature by the administrator. To claim this, the voter opens the number r_i which he used in the blinding technique, and requires the administrator to show the voter's signature. By opening r_i the message e_i is fixed, so the signature which the administrator must show is determined. If the administrator is honest, he can show the signatures for all requests. However, when he dummy voted, there remain the ballots which were not shown in the signature because he cannot forge the digital signature scheme. The fraud is detected here. $\triangle$

11.2.4 Problems with the scheme

The main problem with this scheme is already pointed out by the authors in the previous proof, namely if a voter abstains from voting (i.e. he registers with the administrator but doesn't send his ballot to the counter), the administrator can forge a ballot and use that voter's ID and signature to cast a vote for himself. This fraud is only detectable by the voter who didn't send his ballot.

In effect every voter must have sent in his ballot for the voting process to be trustable. Furthermore, this voting scheme is not receipt free. This means that every voter has a proof of his vote, meaning his vote can be bought by a third party.

11.2.5 Conclusion

Although the authors of this scheme solves most security issues, in our opinion it relies too heavily on the assumption that every voter participates. Therefore we do not consider it fit for use in practise.

11.3 Example of a mix-net based voting scheme

In this section we will discuss the voting scheme proposed by Furukawa et al. [FMM$^+$02]. Whereas the voting scheme discussed in 11.2 only spoke of a theoretical anonymous channel to protect a voter's privacy, the mix-net voting scheme implements this channel. furthermore, the authors give a proof method which can show the honesty of the parties involved in decrypting the votes.

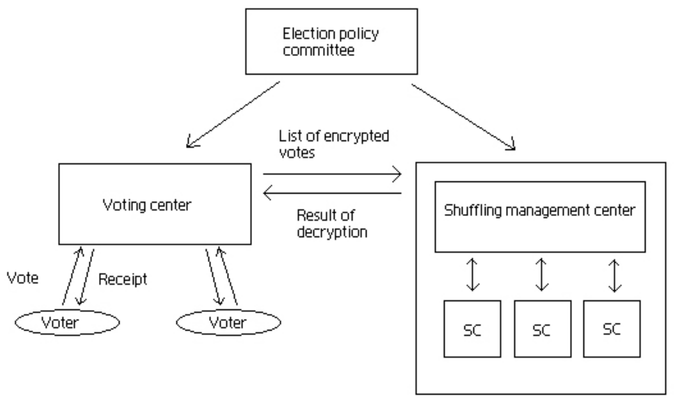


Figure 11.1: Schematic of the mix-net voting scheme

11.3.1 Voting model

The following parties are involved in the voting scheme:

- The voters
- A voting center: collects votes and sends them to the shuffling management center
- A shuffling management center: manages the shuffling of the votes
- Shuffling centers: each center shuffles and decrypts votes
- The election policy committee: decides election policy and verifies the actions of the shuffling centers. It also determines the parameters used in the ElGamal encryption of the votes.

The scheme achieves voter anonymity by distributing the decryption key over the shuffling centers. Voters send in signed and encrypted votes, and receive a receipt of their vote. Votes are processed by the shuffling centers who in turn shuffle and decrypt the votes, then send their output to the next shuffling center. After all shuffling centers are finished they output a decrypted vote, but this vote cannot be traced back to the voter due to the shuffling. The scheme provides shuffle-and-decrypt proofs with which anyone can verify that the output is the result correct decryptions of the shuffled valid votes. See Figure 11.1.

11.3.2 Details of the model

Votes are ElGamal encrypted, with public keys (p, q, g, Y) and secret key $\bar{x} \in \mathbf{Z}_q$, where $Y = g^x \bmod p$. Here, p and q are two primes with $p = kq + 1$, where k is an integer, and g is an element that generates a subgroup $\mathbf{G}_q$ of order q in $\mathbf{Z}_q/\mathbf{Z}$. The vote m_v is encrypted to a cyphertext (G, M) where $G = g^r$ where r is a random element in $\mathbf{Z}_q/\mathbf{Z}$ chosen by the voter. Such cyphertext can be decrypted by a person who knows the secret key $\bar{x}$. However, the secret key is distributed over several shuffling centers, where each center SC_i has a secret key x_i. The sum of their secret keys gives $\bar{x}$.

The shuffling centers each first shuffle the total list of encrypted votes they receive before (partially) decrypting them, and sending them on to the next shuffling center. The shuffle is simply a random permutation of the list of votes. Once the last shuffling center has decrypted the votes, the output is a plaintext list of votes which is sent to the voting center for counting.

For further details we refer to [FMM+02].

11.3.3 Conclusion

The voting scheme based on a mix-net discussed in this section has the feature that it ensures voter privacy. The results of the voting are verifiable using a correctness proof of the shuffling centers. However, this scheme also suffers from not being receipt free, which means votes can be bought.

11.4 E-Voting without 'Cryptography'

In the blind signatures scheme by Fujioka et Al. [FOO93], we have already seen some nice properties such as in 11.2. However, the scheme and all computer based voting schemes rely on software to interact with the voter in order to accept her vote. The easiest way for an attacker to attack a voting system is to attack the software used to vote. By altering the software an attacker could alter or omit undesired votes. Naturally, this is a serious limitation if we would vote over the internet through our own home PCs.

In the paper: E-voting without 'Cryptography' by Malkhi, Margo and Pavlov a voting scheme is introduced that does not require any conventional cryptography on the users' side. This results in the fact that the software being used to vote does not need to be trusted. This is achieved by basing the scheme, the enhanced check vectors protocol, which in turn is based on two already existing tools: *check vectors* by Rabin and Ben-Or [RBO89], which allows one party to hand another party a secret that is guaranteed to be accepted by a group of verifiers, and *anonymous multi-party computation* (AMPC) by Malkhi and Pavlov [MP01], in order to provide the user with a cryptography free receipt when she votes. First, we'll discuss the general outline of the voting scheme, and then we'll go into more detail about the tools used and the protocol itself, followed by the requirements this protocol meets or violates.

11.4.1 The general voting protocol

The voting protocol is composed of four phases:

1. Registration - During this stage a voter (Intermediary) identifies herself to several registrars (Dealers) using a ID such as their passport. A voter receives her vote

vectors for each possible ballot s. Such a vote vector corresponds to the secret V that a voter receives from the registrar. Talliers (Receivers) receive check vectors B that correspond to the vote vectors in such a way that $VB = s$.

2. Testing - During this stage the voter verifies that the vote vectors she receives from the registrars would indeed enable her to vote.
3. Voting - The voter sends her vote vector V that represents her desired ballot s to some of the talliers. Each of these talliers can verify with the corresponding check vector that indeed $VB = s$ (So the vote is valid) and sends a receipt back to the voter. With this receipt the voter can verify that her vote was received properly by the talliers.
4. Counting - each tallier publishes all the ballots it receives, and everyone can verify the results.

11.4.2 Check Vectors

Rabin and Ben-Or describe a fairly simple process. This process will enable the voting scheme to carry out authentication of information. Although it is not as powerful as digital signatures, the advantage of these check vectors are that it needs no cryptographic assumptions, and is thus ideal for the voting protocol. Basically, there are 3 participants in the process: 1. The dealer D (registrar), who holds data s (the voting ballot), 2. *INT* - an intermediary (the voter), who receives s from D, and who at a later time may wish to pass it on to, 3. *RCV* - the recipient (the tallier). The *RCV* is said to accept s, if he believes that this value originated with D. The protocol of acceptance has the following properties:

1. If D and *RCV* are honest, then *RCV* will always accept s if it actually originated with D, and will reject with probability $\geq 1 - \frac{1}{2^k}$ any other value *s'*. k is the security parameter.
2. Regardless if D is honest or not, *INT* will know, with probability of mistake $\geq \frac{1}{2^k}$, whether *RCV* will accept the s that he holds.

For all the computations it is assumed that a large enough prime, $p > 2^k$ is chosen with $s \, \epsilon Z_p$.

The dealer D chooses two random numbers $b \neq 0$ and y both in Z_p, and hands to *INT* the pair (s, y). D computes $s + by = c$. Then the dealer hands to *RCV* the vector (b, c) which will be known as the Check Vector. Later, when *INT* will transmit (s, y) to RCV, he will compute $s + by$ and will accept only if it equals c. It is easily seen that this check vector method essentially disarms a dishonest *INT* from the ability to pass to *RCV* a false value, s, which *RCV* will accept.

The proof that the above properties hold will not be discussed here, but can be found in [RBO89].

Also, this protocol can be extended to the case of several receivers. Let us assume the case where there is one dealer D, one *INT*, and receivers $RCV_1, ..., RCV_n$. The dealer

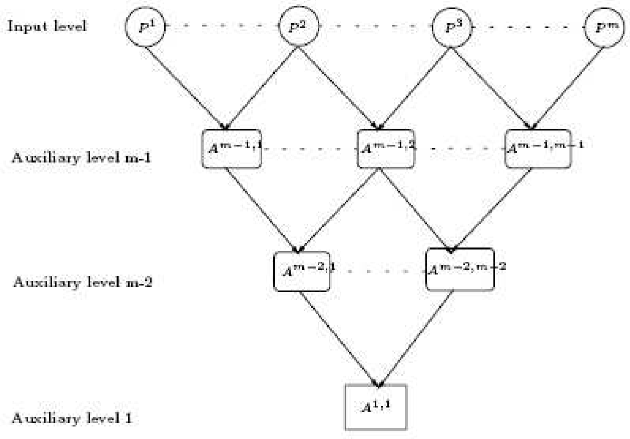


Figure 11.2: THE TRIANGULAR COMMUNICATION GRAPH IN AMPC

hands to *INT* the secret s and random values $y_1, ...y_n$, y_i will eventually be handed to RCV_i to authenticate s. And finally, each RCV_i receives from D a check vector (b_i, c_i) created in the way described above.

11.4.3 Anonymous multi-party computation

Malkhi and Pavlov present in their paper: Anonymity without 'Cryptography' [MP01] a building block which provide users electronic anonymity when using applications. *Anonymous multi-party computation* (AMPC) transforms a list of input values into a list of output values such that the output values cannot be correlated via the computation to any input values. Like the check vectors, AMPC is designed in such a way that users are not required to employ conventional cryptography.

The implementation is derived of the mix-net (shuffle) protocol by Chaum and is constructed as follows. A system of players (see Figure 11.2) is built that performs a multi-party computation. Every message between the players is sent through a secure channel so that the secrecy of the messages is maintained. There are m players as input players. These players receive initial values at the beginning of computation. They then apply any homomorphic function which they have agreed upon to the input values in order to produce permuted output values corresponding to these inputs, such that no correlation is revealed between the inputs and the outputs. This is achieved by employing m auxiliary levels. They then split and send these values to players of the next level. Between every level a permutation chosen by the players of that level is applied until at the final level all results are published. In the voting scheme however it is necessary that the outcome will be kept secret. This can be done by employing a square network with m players at each level (see Figure 11.3). The computation proceeds the same, but the side players now split the values the same way as the middle players do. The bottom level produces the split pieces of the desired output and sends

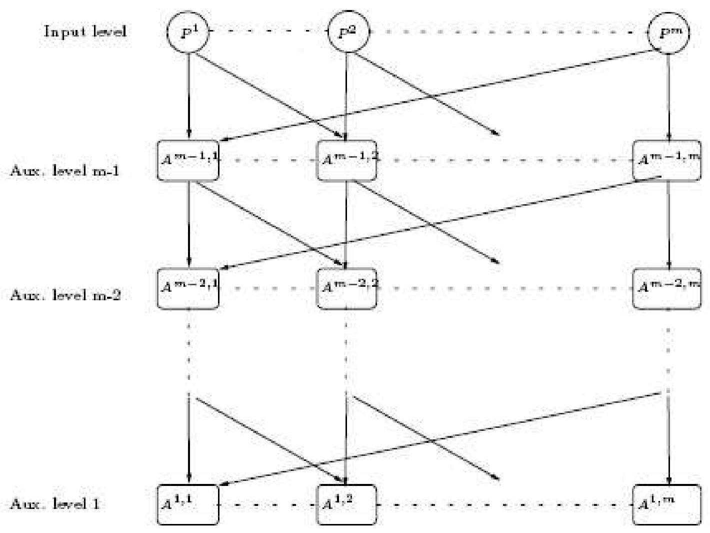


Figure 11.3: THE SQUARE COMMUNICATION GRAPH IN A SECRET PRESERVING AMPC

it over secure channels to the target recipient.

11.4.4 The enhanced check vectors protocol

We have now discussed the two basic tools the protocol will be based on. However, the original check vectors protocol does not yet meet all the requirements yet. The INT is able to forward the secret to multiple RCV, but needs to use a different random value y_i for each RCV_i. Translated to the voting scheme this would mean that the voter (INT) has to send a different ballot to each potential tallier (RCV_i). This clearly is not a very suitable scheme. Therefore we extend the check vectors protocol to be able to send the same secret V with a meaning s to one or more receivers. Each one of the receivers should be able to independently verify the secret meaning and the fact that the secret originated from a dealer, but a subset b (the security threshold) or less of these receivers should not be able to reconstruct the secret V. Also in the basic check vectors protocol, only one dealer sends the secret V to INT. The protocol has to be extended in such a way that several dealers will construct the secret V, and in such a way that a subset b or less dealers should not be able to reconstruct the secret V. Moreover the vote and check vectors cannot be correlated to the voters, as that would give away their identity. This means that the Dealers cannot be allowed to compute the check vectors and send them to the receivers. This is solved by using the AMPC. The idea is to generate each coordinate of the secret V jointly by the dealers out of which the vote vectors are made by the voters themselves. Further, each dealers sends his information about the vote vector as input to the AMPC, which produces output that is no longer correlated to the vote vectors. This output can now be used by a trusted third party to compute the check vectors and send them to the receivers.

The protocol is now complete and below the enhanced check vectors protocol is given:

1. For each INT the *j*'th registrar issues additive shares $v_{1j}, ..., v_{nj}$ of each of the n coordinates of vector V. For purposes of correlating V with its check vector, the *j*'th dealer also issues an additive share vid_j of a voter-id vid.

2. The voter can now compute the coordinates of her secret vector V and her voter-id vid by summing up the shares received from the m different registrars $v_i = \sum_{j=1}^{m} v_{ij}$ and $vid = \sum_{j=1}^{m} vid_j$.

3. Now, an AMPC is employed as follows: for each coordinate i and all voters, the j'th dealer provides v_{ij} and vid_j as input to the j'th AMPC top player. The AMPC then produces a permuted list of i'th coordinates v_i of all voters along with their corresponding vid. This list is sent to a third party. This party has now no correlation between the pairs $\langle v_i, vid \rangle$.

4. The trusted third party can now compute check vectors B for V such that $BV = s$ and sends these along with the vid to the talliers

At the end of this protocol, each tallier holds a list of check vectors $\langle vid, B \rangle$. that each corresponds to an anonymous voter holding $\langle vid, V \rangle$ such that $VB = s$.

11.4.5 The voting protocol

We can now present the complete voting scheme using the enhanced vector checking protocol:

1. **Voter registration**: The enhanced vector protocol discussed in the previous section is used by the group of registrars to issue all voters who have identified themselves to them their anonymous secret vectors V, secret meaning s, secret voter-ids vid, and to issue corresponding check vectors to talliers (via a third party). For each ballot s (the candidate), each voter gets several pairs of vote vectors $\{\langle V_{1,0}, V_{1,1} \rangle \langle V_{2,0}, V_{2,1} \rangle, ..., \}$ in order to facilitate testing and receipts.

2. **Testing**: In this stage the voter verifies that talliers indeed hold a valid check vector for her vote. As stated above each voter holds several pairs of vote vector for each candidate. She randomly selects one of the vectors, say $V_{1,0}$, and sends is to several talliers over an anonymous network, this could again be done via AMPC. Each tallier sends back the check vector corresponding to the adjoining vector in the pair, in this case $B_{1,1}$. The voter can now verify with $V_{1,1}$ that $V_{1,1}B_{1,1} = s$. She should now publish the vote vector pair used for verification to disqualify it for use in the elections. The tallier can verify the check vector indeed came from a honest registrar.

3. **Voting**: Now basically the same as in the testing phase happens. The voter sends one vector who represents her vote along with her vid to several talliers through

AMPC. Each tallier verifies that this vote vector is valid and sends the adjoining check vector back to the voter, who can use this check vector as a receipt.

4. **Counting** At the end each tallier publishes each vote vector it received along with its *vid*. Other tallier who did not receive the vote vector from the voter can verify the validity, as they have received the same check vectors. The voter verify if her vote appears on the published list. Of course, the majority of the votes then wins the elections.

11.4.6 Security analysis

The voting protocol meets quite a number of requirements given in section 11.1. Below we will discuss where its strengths and weaknesses lie.

1. **Completeness** The voting system must not allow ballot stuffing by any participant, and prevent the elimination and modification of valid ballots. The number of registrars, auxiliaries and talliers is each m, out of which a total of $\lceil \frac{m}{2} \rceil - 1$ corrupt authorities combined of all types is assumed. We will not discuss the proof here, but can be read in the paper.

2. **Eligibility** That a valid voter receives a valid vote is guaranteed by the testing phase. The probability of detecting corruption at the registration phase can be made as high as possible by the amount of pre-voting testing.

3. **Unreusability** That the voter can only vote once, is easily achieved by only allowing its *vid* once on the published list given out by the talliers.

4. **Verifiability** We have individual and universal verifiability. Individual verifiability is easily achieved. Because of the receipt they get from the tallier, voters can easily check if their votes have been counted correctly. Universal verifiability is achieved by publishing the list of vote vectors with their corresponding voter ids. Every tallier can check if all votes all valid.

5. **Privacy** Every vote is kept secret. In order to link the voter with his vote, one must correlate the voter with the voter id or the vote vector. Since Registrars only see a share of the vote vectors and voter id, and the talliers get ballots over an anonymous network, neither of them is able to correlate the voter with its vote vector or id. The third party do know whole vote vectors. They receive them however through AMPC, so they cannot correlate them to any input share which registrars can relate to voters.

6. **Receipt-Freeness** This requirement is actually in contradiction with individual verifiability. When a voter is able to verify how she voted, it also the case she can proof how she voted. In this case she can give up her voter id in order to proof how she voted.

Although this protocol does not meet the requirement of receipt-freeness it has some very interesting properties, the main property being the fact that the software being used does not have to be trusted. This is a very interesting property if we would want to vote over the internet one day.

11.5 Conclusion

We have seen a few voting protocols who implement most of the requirements we set out in section 11.1. However, none of the protocols meet all of the requirements, so there is basically no protocol available that is perfect for electronic voting. But in any way, these protocols have certainly some advantages such as verifiability and voter privacy over the current voting processes. Maybe a future protocol will be able to contain these advantages and include the requirements not yet met, to make electronically voting over the internet possible. It is however the case that although the voting protocols meet the requirements of privacy, voting over the internet can lack privacy on the physical side, such as a watching spouse or other parties who can put the voter under pressure. That is why we think voting over the internet that would decide major elections will never be successful. However, implementing these schemes in electronic voting machines that are used regionally and are under the supervision of neutral parties could become a reality one day. For further discussion of the reality of electronic voting, see [GNR$^+$01].

Chapter 12

On Robust Combiners/Tolerant Constructions

Written by *Sjoerd van Hagen.*

This chapter tells you something about robust combiners, also called Tolerant Combiners, a very interesting subject in cryptography. There is not a lot of material on it yet but it is a very nice way to improve security for cryptographic problems. The by far largest part of information in this chapter can be found in [HKN$^+$05] and [Her05].

Note that I use two terms for the same thing, Robust Combiner and Tolerant Construction. In [HKN$^+$05] it is called Robust Combiner and in [Her05] a Tolerant Construction. I am not in favor of one of the two so I will use them both, maybe in the future one name will become favorable.

12.1 Introduction

The security of a lot of algorithms in Cryptography depends on the assumptions that some functions cannot be computed efficiently, for example the discrete logarithm and prime factoring. The latter assumption implies that computing roots modulo a composite number is also infeasible. However, these assumptions where never actually proven. Luckily, the probability of the existence of an algorithm to compute these functions efficiently is very small.

What if someone would succeed to find a function that can compute one of these functions efficiently? In that case it would be nice to have an extra lock on the door. This is where robust combiners kick in. A robust combiner combines multiple cryptographic algorithms into one solution for a cryptographic problem. It is secure if at least a fixed number of these approaches, are secure. This concept is informally expressed in definition 12.1.

Definition 12.1 *A (k,n)- Robust Combiner for a cryptographic primitive $\mathcal{P}$ is a construction that takes n candidate schemes for $\mathcal{P}$ and combines them into one scheme such that if at least k of the candidates securely implement $\mathcal{P}$ then the combiner also securely implements $\mathcal{P}$.*

Other reasons for using Robust Combiners are for example:

- Increasing the security parameter, for example the key size.
- Using implementations or machines from sources which you do not entirely trust.
- Avoiding bugs in software by combining several versions hoping that at least k are correctly implemented.
- If one of the primitives fails it can be replaced by a previously unknown stronger primitive so fast that an opponent cannot really exploit the weakness.

Theoretically speaking Robust Combiners can be very useful tools in constructions and reductions between cryptographic primitives. This is for example the case when it is guaranteed that a certain construction exists at all. The nice thing here is that if you can prove that of a number of candidates one must work you can combine them in such a way that the result always works. Still you do not know which of the candidates actually is the right one. If for example you can prove that of two functions one will work if the other doesn't and vice versa, combining the two functions in the right way will give you a function which always works well.

- Levin introduced an Universal-one way function which is guaranteed to be an one way function if one way functions exist at all.
- Hastad has shown that if a polynomial number of candidates for pseudo-random generation are given of which one is guaranteed to be a pseudo-random generator, the simple XOR of the outputs of there candidates is guaranteed to be a pseudo-random generator.

12.2 Robust Combiners in detail

In this section we will have a closer look at Robust Combiners. We will define Robust Combiners and their properties more precisely starting with giving a definition of a cryptographic primitive in 12.2. We use the notion of a probabilistic polynomial Turing machine(PPTM).

Definition 12.2 (Cryptographic Primitive) *A Primitive $\mathcal{P}$ is a pair $\langle F_{\mathcal{P}}, R_{\mathcal{P}} \rangle$ where $F_{\mathcal{P}}$ is a set of functions $f : \{1,0\}^* \mapsto \{1,0\}^*$, and $R_{\mathcal{P}}$ is a relation over pairs $\langle f, M \rangle$ of a function $f \in F_{\mathcal{P}}$ and a machine M. The set $F_{\mathcal{P}}$ is required to contain at least one function which is computable by a PPTM. A function $f : \{1,0\}^* \mapsto \{1,0\}^*$ implements $\mathcal{P}$ or is an implementation of $\mathcal{P}$ if $f \in F_{\mathcal{P}}$. An efficient implementation of $\mathcal{P}$ is an* *implementation of $\mathcal{P}$ which is computable by a PPTM. A machine M $\mathcal{P}$-breaks $f \in F_{\mathcal{P}}$ if $\langle f, M \rangle \in R_{\mathcal{P}}$. A secure implementation of $\mathcal{P}$ is an implementation of $\mathcal{P}$ such that no PPTM $\mathcal{P}$-breaks f. The primitive $\mathcal{P}$ exists if there exists an efficient and secure implementation of $\mathcal{P}$.*

The set $F_{\mathcal{P}}$ describes the functionality of the cryptographic primitive. If for example we consider encryption the set $F_{\mathcal{P}}$ should be constructed in such a way that if a message m encrypted with key k is decrypted with the same key k it recovers the same message m. If we consider a primitive for Key Agreement, we would like that all parties in the agreement will end up with the same key.

The set $R_{\mathcal{P}}$ describes the security of the primitive. If we take for example RSA, which is based on the assumption that prime factoring is infeasible, the set $R_{\mathcal{P}}$ would contain among others:

- Trying to divide by all (prime) numbers between 0 and $\sqrt{n}$ where n denotes the number to be factored.
- Pollards Factoring algorithm.

As long as the algorithms in $R_{\mathcal{P}}$ cannot be implemented by a PPTM the primitive is considered secure. Encryption and decryption are polynomial in the size of the key and the size of the messages. Attacking is considered not to be possible in time polynomial in the size of the key. This does not mean that all implementations of RSA are secure however, a lot depends on the chosen key size, but this is something our definition does not take into account.

Definition 12.3 ((k, n)-Robust Combiner) *Let $\mathcal{P}$ be a cryptographic primitive. A (k, n)-Robust Combiner for $\mathcal{P}$ is a PPTM that gets n candidate schemes as inputs, and implements $\mathcal{P}$ while satisfying the following two properties:*

1. *If at least k candidates securely implement $\mathcal{P}$ then the combiner also securely implements $\mathcal{P}$.*
2. *The running time if the Combiner is polynomial in the security parameter m, in n, and in the lengths of the inputs to $\mathcal{P}$.*

This definition is very broad. A Robust Combiner could even ignore all candidates and implement the primitive itself. We could like to be able to use a more restrictive definition to be able to prove claims we cannot prove for Robust Combiners as defined above. Therefore we define the notion of a black-box combiner.

Later on in this chapter we will be using the term Tolerant Constructions as well. Also the context will change a little when we are talking about Tolerant Constructions. We assume that all Tolerant Constructions are constructions using two primitives unless stated otherwise. We use three terms to indicate the security of Tolerant Constructions:

1. 1-Tolerant: the construction is secure if at least one of the primitives is secure.
2. 0-Tolerant: the construction is secure if both primitives are secure.

3. NOT-Tolerant: the construction is even less secure than the least secure primitive it uses.

Definition 12.4 (Black-Box Combiner) *A $(1,2)$-Robust Combiners is said to be* ***black-box*** *if the following conditions hold:*

1. Black-box implementation: *The combiner is an oracle PPTM given access to the candidates via oracle calls to their implementation function.*

2. Black-box proof: *For every candidate there exists an oracle PPTM R^A (with access to A) such if adversary A breaks the combiner, then the oracle PPTM R^A breaks the candidate.*

So what does this mean? The combiner can make calls to its candidate schemes but does not have access to the execution of the candidates. Basically it gets only the output of the execution and no extra information.

12.3 Examples of simple Robust Combiners

12.3.1 Encryption

Robust Combiners have been around for quite some time but the term 'Robust Combiner' is not often encountered in literature. If you interpret the term 'Robust Combiner' in a broad way encrypting multiple times with the same key is can also be seen as a Robust Combiner. This only offers extra protection because of the increased key length and in case of bad choice or key management by the owner.

One of the most used examples of this principle, namely 3-DES, cannot be seen as a Robust Combiner. If one of the DES encryption keys is lost or guessed by Oscar, only 2 DES encryptions remain. 2-DES is not considered secure, it could be broken using a meet-in-the-middle attack. 3-DES would therefore be a $(3,3)$-Combiner, 3-DES is secure if all three keys are secure, so it is not robust at all.

A better example of a Robust Combiner for encryption would be to encrypt with algorithm A and encrypt the result again with algorithm B, which use a different key and a different design. In this case the secret is safe while at least one of the encryption algorithms is secure. This principle can be generalized to a $(1,n)$ Robust Combiner by adding more encryption algorithms to the sequence. The hardest part would be to select algorithms which are fundamentally different from each other, for example DES and IDEA are based on the same principles and both admit a differential attack.

12.3.2 One way functions (OWF's)

One Way Functions have a very simple Combiner if you look at it in a naive way. We will do that for now, later on we will reveal the error and solve it using another combiner. Simple function composition seems to be a Combiner for One way Functions.

It is easy to see that if one of the functions is one way, the composition is one way as well. All insecure functions applied after the one that is secure can be reversed to find out the output of the one secure function. Because it is impossible to find the input to the one secure OWF one cannot find the output of the functions before the OWF and hence cannot find the input to the combiner. There is however a little pitfall which we will encounter later on in this chapter.

A nice thing of One Way Functions is that they are equivalent to many other cryptographic primitives. Equivalency of primitive means that one primitive is reducible to and from another primitive. Primitives that are equivalent to One Way Functions are semantically secure private key encryption, pseudo-random number generators, functions and permutations, digital signatures and bit commitments.

Combining these functions is done by first reducing all candidates for the primitive $\mathcal{P}$ to One Way Functions, combining them to a Robust Combiner implementing One Way Function and reduction back to the primitive $\mathcal{P}$. Usually reduction to and from OWF is not preferable because the primitives can be combined directly. For example if you want to combine pseudo-random number generators, just take the XOR of the outputs of the candidates. If all candidates are independent of each other and one of them truly is a pseudo-random number generator the combiner will be a pseudo-random number generator as well.

12.3.3 Key Agreement

Two parties want to agree on a key, however from the communications an attacker must not be able to find out anything about the key. Both parties should of course get the same key in the end otherwise they cannot communicate with each other. A common method for Key Agreement is the Diffie-Hellman protocol. Let us apply the definition of a cryptographic primitive.

The set $F_{\mathcal{P}}$ would contain a function f which would be computed as follows:

1. First Alice and Bob choose a public number g.
2. Alice and Bob both choose a secret number x_A and x_B.
3. Alice computes public number $p_A = g^{x_A}$ and sends it to Bob. Bob computes public number $p_B = g^{x_B}$ and sends it to Alice.
4. Alice takes Bobs public number and calculates key $k = {p_A}^{x_A}$. Bob does the same with Alice's public number and his secret number.
5. Alice and Bob use k to communicate.

Of course Alice and Bob will get the same key so the functionality of the primitive is accounted for. The set $R_{\mathcal{P}}$ contains functions to compute the discrete log because that is what the adversary has to do to get the key. Because discrete logarithm is assumed infeasible this primitive is considered secure.

However the security is not guaranteed so if we can we would like to find a Robust Combiner for increased security. There are two cases that can be distinguished, in the first case we can safely assume that the functionality of the used primitives is working correctly. This is a legal assumption in most cases because the functionality usually can be proven quite easily. In the second case we review a Robust Combiner which does not make the assumption of correct functionality and try to arrive at a correct key agreement with very high probability.

The first case is very easy because we know that the outcomes of all used primitives on Alice's side are equal to the outcomes on Bobs side. Simply computing the XOR of all outcomes will give a key k which is equal on both Alice's and Bobs side. If an attacker will be able to crack $n-1$ of the primitives he would still not know anything about k because every key k is still possible and equally likely as well. Therefore we can conclude that this Robust Combiner is secure whenever one of the primitives used is secure.

A more difficult challenge is to create a Robust Combiner in cases in which the functionality of the primitives is not proven to be correct. First we have to be able to say something about the probability of an error occurring. If a primitive is wrong in every execution we will be unable to a Robust Combiner of course.

To compute an assessment of the probability of a wrong outcome each party computes n^2 executions of the primitive. So both Alice and Bob simulate n^2 executions by playing the roles of both players of the primitive and only accept the primitive if the candidate always computes the same key for both players.

We denote by x the random inputs of a candidate:

$$X_{bad} = \{x \mid \text{ candidate } f \text{ fulfills functionality when running } x\}$$

Then we have that:

$$\Pr[\text{ candidate } f \text{ passes test } \mid \Pr(X_{bad}) > \frac{1}{n}] \leq (1 - \frac{1}{n})^{n^2} \leq O(2^{-n})$$

So if a primitive does pass the test we know that with a probability of $O(2^{-n})$ it will give the correct outcome with a probability of $1 - \frac{1}{n}$. Harnik et al[HKN$^+$05] claims that the use of error correcting code can decrease the probability of Alice and Bob ending up with a different key to a negligible small value.

12.4 General construction of Tolerant Constructions

The paper by Herzberg [Her05] is trying to create basic constructions which can be applied when the problem at hand has certain properties. These constructions can be divided in cascade or parallel constructions. They also try to prove that so called folklore constructions, like cascading encryption schemes, are tolerant constructions.

12.4.1 Cascade constructions

Cascade constructions combine two cryptographic primitives by means of function composition, probably the most straightforward construction one can think of. We have already seen this for encryption and for one way functions. In this subsection we will go into more detail on this construction and correct our naiveness on the combiner for one way functions.

For encryption, cascade combiners are proven to be secure in case of reasonably likely attacks. In case of a unreasonably unlikely attack, namely the adaptive chosen ciphertext it does not improve security. Proof of these claims can be found in the paper [Her05].

One Way Functions

As we have seen One Way Functions seem to admit robust combiners by function composition. Herzberg shows that this is not true unless some requirements are met. He shows by counterexample that cascade is tolerant if the possible outputs of each primitive is a permutation of the possible inputs for the next primitive.

Assume $h(x)$ is an One Way Function. He then defines one way functions f and g, where $g(x)$ equals $h(x)$ padded with a string of zero's and f equals 0 if the input ends with a string of zero's, else it equals $h(x)$. Both functions are one way functions but the composition clearly always outputs 0 and thus does not meet the specifications of a one way function, which will see later on when considering a parallel construction of OWF.

This seems like a very artificial example but note that if each candidate has a probability of collision p_i and the candidates are not related, the probability of collision of the Combiner equals $\sum_{i=1}^{n} p_i$, which can get large. When looking at hash functions this problem is very relevant, because of the compression a hash function is supposed to perform on the input. So for hash functions this is a very bad combiner.

Commitment

A commitment scheme is a cryptographic primitive in which Alice sends a message to Bob without revealing the content of the message. The commitment can be revealed by Alice only. Because Bob actually has the commitment Alice cannot alter the content of the message. At the end of the protocol Alice gives Bob a key which he can use to get to the message and to verify whether this really was the message Alice had committed. Think of it as a message in a locked box sent by Alice to Bob, to which Alice has the key and gives it to Bob at the end of the protocol.

A commitment scheme can be perfectly binding, meaning it is impossible for Alice to change the commitment after she made it. A commitment scheme can also be perfectly concealing, meaning it is impossible for Bob to find out the message in the commitment without Alice revealing it.

A commitment can never be perfectly binding and perfectly concealing at once. If it is perfectly binding there can be exactly one value for the message given a certain commitment and therefore it is not perfectly concealing. If it is perfectly concealing all messages have to be possible given a commitment and therefore it is not perfectly binding. So the security of a commitment scheme is always a computational one. Therefore it would be nice if we would have a Tolerant Construction for this problem.

Now let us apply the cascade combiner. Alice takes two commitment schemes and applies the first to the message and the second one to the commitment value given by the first function.

Lemma 12.5 *Cascade commitment is 1-Tolerant for the hiding specification, and 0-Tolerant(preserves) the binding specification.*

The binding specification is preserved meaning that the binding is securely implemented if both primitives securely implement the binding. Cascade commitment is 1-Tolerant for concealment, or hiding, meaning that the concealment is securely implemented by the combiner if one of the primitives is securely implemented.
Proof. Assume we cascade two commitment schemes c_1 and c_2 into one commitment scheme c. Our new commitment function would be $c = c_1 \circ c_2$. Now if Bob wants to find the message he has to invert both functions c_1 and c_2. So if one of them is secure he cannot thus this construction is 1-Tolerant for hiding.
However if Alice wants to change the message she can either try to find a message which creates a collision in outputs of c_2 and consequently in c or creates a collision in the output of $c_1 \circ c_2$. So both commitment schemes have to have a secure binding property. $\triangle$

To make sure that the probability of collision does not increase with the number of candidates used, like we have seen with the cascade combiner for one way functions Alice has to send all intermediate values of the computation and all corresponding decommitment values. Bob can now verify that Alice is playing fair. Drawback of this construction is of course increase in de length of de decommitment string.

Signature schemes

A cascading scheme $s = s_1 \circ s_2$ for a signature would be to sign the signature given by s_2 with scheme s_1. To be able to verify the signature the intermediate signature has to be sent with the message as well.

Lemma 12.6 *The cascading construction for Signature schemes is 0-Tolerant for existential unforgeability and 1-Tolerant for universal unforgeability.*

Proof. Let s_1 be the first signature and s_2 the signature of s_1. If Oscar can find a message with the same signature s_1 as one of the messages Alice has already sent (existential forgery) he can simply take the s_2 of that message as well to make a forgery. However if he wants to be able to sign any message he likes (universal forgery) he will have to create his own s_1 and s_2 and thus breaking both candidates. $\triangle$

12.4.2 Parallel combiners

Parallel combiners all take some input and independently compute an output which is then combined into one output of the combiner. The input to a primitive is usually one of the following: the whole input to the combiner, part of the input to the combiner or some combination of inputs, often by XORing or secret-sharing. The output can be put together by simple concatenation, XORing or secret-sharing of the individual outputs.

Split-Parallel-Concat

One of the easiest parallel constructions available is Split-Parallel-Concat which, like the name says, first splits the input, then calculates the outcomes of the primitives on the parts in parallel and concatenates the outputs into one output. In case of the One Way Function the input will be cut into pieces of equal size. Every piece is input to a candidate one way function. The outputs are concatenated into the output of the combiner.

Lemma 12.7 *The Split-Parallel-Concat (sc) construction is tolerant for OWF specifications.*

I will prove this using the definition of a strongly one way function given below.

Definition 12.8 *A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is called (strongly) one-way if the following two conditions hold:*

1. *Easy to compute: There exists a (deterministic) polynomial-time algorithm, A, so that on input x algorithm A outputs $f(x)$ (i.e., $A(x) = f(x)$)*
2. *Hard to invert: For every probabilistic polynomial-time algorithm, A', every polynomial $p(\cdot)$, and all sufficiently large n's*

$$P(A'(f(U_n), 1^n) \in f^{-1}f(U_n)) < \frac{1}{p(n)}$$

Proof. If all candidates but one can be reversed the probability of A' finding a reverse to the combiner is still smaller than $\frac{1}{p(\frac{n}{k})}$ for every polynomial, where k is the number of candidates. We just have to take care and make sure our value of $\frac{n}{k}$ is large enough. The proof for weak one way functions is similar to this one and is skipped. △

Copy-Parallel-Concat

Copy-Parallel-Concat is very straightforward as well. The input to the construction is being copied and used as input to all used primitives, at the end the outputs are again being concatenated. It is widely used for different purposes, for example hash functions, signatures and commitment. We will look at an example for commitment only but make statements about the other primitive as well.

Again the application of this construction on commitment is very easy. Alice will compute the commitment values c_1 and c_2 by applying both candidates on the message. She will send both c_1 and c_2 to Bob. At the end of the protocol she will send two decommitment values to Bob. Bob will check if both commitments are valid by using the decommitment values.

Lemma 12.9 *The Copy-Parallel-Concat construction is 1-Tolerant for binding and 0-Tolerant for hiding.*

Proof. Alice will have to forge the message such that both commitments are still valid. If at least one of them is secure this is not possible therefore this construction is 1-Tolerant for binding. Bob only has to break on of the candidates to be able to recover the message therefore this construction is 0-Tolerant. △

This construction is also useful in other situations, mentioned below.

1. 1-Tolerant for integrity specifications of hash functions.
2. 1-Tolerant for existential and universal unforgeability of signature schemes.

Note again that the output string can be very long which is a serious drawback of this method.

Care must be taken not to look at this construction as a means of increasing the security by increasing the length of the output. We will look at this construction applied to hash functions to show that this is not as secure as having a secure hash function of twice the number of bits. Actually this was thought to be secure for a long time until it was proven wrong by Antoine Joux [Jou04] as shown in the next paragraphs. Note that he calls this construction cascading but it actually corresponds to this parallel construction.

Combining hash functions F and G of a message M should ideally be as secure as a hash function of a bit length of the sum of the bit lengths of F and G. Now finding 2^{n_g} collisions for F where n_g is the number of output bits used by G is not much harder than finding one collision because of the birthday paradox argument. To make this more intuitive consider the birthday paradox, when even more people enter the room the probability of pairs having the same birthday increases with every person who enters. Of all these 2^{n_g} collisions it is very likely that there will be at least one with a collision in G.

This shows that concatenation of hash function outputs does not increase the security very much. However it does protect against attacks when one of the hash functions is flawed. If one of the hash functions has an output which is not uniform for instance, one can more easily create a collision. Luckily there is a second hash function to back it up. So this combiner can still be useful as long as you keep in mind that the security of the combiner is that of the hash function with the smallest number of output bits (because we do not know which of the functions may be flawed).

XOR-Parallel-Concat

Asmuth and Blakley [AB81] proposed this combiner to be used for encryption. The construction works as follows:

1. Compute $xor = m \oplus r$, where m is the message and r a random string.
2. Encrypt both xor and r using the different candidates.
3. Concatenate the results.

If an adversary now manages to break one of the candidate he will get either the xor or the random string r. He must have both to be able to reconstruct the input message m. Therefore this combiner is 1-Tolerant. A big drawback obviously is the doubling of the output size.

Share-Parallel-Concat

This combiner seems a little bit more complicated but is very simple as well. It is a tolerant construction for commitment with some very nice properties. The input message is divided using a secret sharing scheme in this case Shamir's scheme. In this scheme the number of shares that do not give any information about the input can be set to any value t between one and the number of shares. The minimum number of shares to reconstruct the message is $t + 1$.

The shares are fed to the commitment candidates which all produce a partial commitment. These partial commitments are being concatenated and sent to Bob. When Bob is allowed to see the message Alice will send the concatenated partial decommitment values to Bob. Bob will reconstruct the message and thus ending the protocol.

Lemma 12.10 *The Share-Parallel-Concat construction of* $2t + 1$ *commitment schemes is* t*-Tolerant for binding and hiding for every* $t \geq 1$*.*

Proof. Because t shares do not say anything about the secret Bob would have to break $t + 1$ candidates to cheat Alice and find out the content of the message. Alice on the other hand would have to forge $t + 1$ shares to make sure that there are less than t valid shares remaining for Bob to reconstruct the secret. △

Of course it is very nice that we can make this scheme more secure by adding more and more candidate functions. The drawback is the very long length of the decommitment string. Most commitment schemes require that the input message is included in the decommitment. Because shares in Shamir's scheme are as long as the secret message, which can be arbitrarily long, the length of the decommitment string is a multiplication of the length of the message with the number of candidates used.

12.4.3 Composite constructions

By composite constructions we mean constructions which are built of different kinds of simple constructions, for example a Copy-Parallel-Concat construction and a Cascade construction. This can have various purposes for example combining constructions which each are Tolerant for a different property of the primitive it implements. I will show this for Commitment.

We have seen three constructions for Commitment, a cascade construction, a copy-parallel-concat construction and a share-parallel-concat construction. Share-Parallel-Concat is Tolerant for both hiding and binding specifications but is very expensive in the length of the commitment and decommitment schemes. The copy-parallel-concat is Tolerant for binding and the cascade construction was Tolerant for hiding. We can combine these constructions into a construction which is Tolerant for both. Next I will give two examples of such constructions.

Two composite constructions for commitment

D-construction is a combination of the cascade and the copy-parallel-concat construction for commitment. Herzberg named it D-construction after its shape which resembles a D. The construction takes four candidates, the first two and the last two are combined using a cascade construction. The two resulting constructions are being combined using a copy-parallel-construction.

This construction is of course a lot more efficient than the share-parallel-concat construction. It only takes four calls to its candidates but more important, the input message to all of the candidates is the same so it only has to be sent once. Furthermore there are two commitment and four decommitment values, but they usually are a lot smaller than the length of the message.

Lemma 12.11 *D-construction is 1-Tolerant for both hiding and binding specifications of commitment.*

Proof. To make sure that the copy-parallel-concat is secure we must be sure that both cascade constructions meet the hiding specification. So both cascade constructions must have at least one secure candidate. If we would require that 2 candidates have to be secure we must be sure that they are not in the same cascade construction. But than we can have that none of the cascading constructions is secure for binding and we cannot guarantee that the D-construction is secure. So we have to have 2 secure candidates in one of the cascade constructions and at least one in the other. Therefore we require 3 secure candidates, hence the construction is 1-Tolerant. △

There is also a big drawback to this construction however and that is that it uses four candidates for 1-Tolerance where share-parallel-concat uses only three candidates. Luckily this can be easily fixed by using a construction called the E-construction, again after its shape.

The E-construction takes three candidates. Three different pairs can be made out of these three candidates, these are used to build three cascade constructions which are combined using copy-parallel-concat. In this case five candidate calls are needed.

Lemma 12.12 *E-construction is 1-Tolerant for both hiding and binding specifications of commitment.*

Proof. Again to make sure that the copy-parallel-concat construction is secure all of the cascade constructions have to be secure for the hiding specification. If you pick any two of the three candidates to be secure this will be the case because every cascade construction will have at least one of the secure candidates among its candidates. If we have two secure candidates we also have that one of the cascade constructions is secure for binding, hence the E-construction is secure for binding. Because we need two of the three candidates to be secure the E-Construction is 1-Tolerant. △

12.5 Building $(1,n)$-Combiners from $(1,2)$-Combiners

If a primitive allows the construction of a $(1,n)$-Combiner it obviously also allows for the construction of a $(1,2)$-Combiner. The reverse however is not always the case. Of course if one would build a binary tree with as internal nodes $(1,2)$-Combiners and as leaves the primitives one will get a primitive $\mathcal{P}$ which gets the job done when no less than one of the used primitives gets the job done.

However to be qualified as a robust combiner other conditions have to be met. The combiner has to be polynomial in its running time. The height of the tree is logarithmic in n and if the running time of $(1,2)$-Combiners is m times that of its candidates the running time of the whole construction is $m^{\Omega(\log k)}$. To have a polynomial running time m must be a constant.

Let us try to make this a little bit more intuitive. If you think of the tree, all internal nodes are $(1,2)$-Combiners. They all take $O(1)$ time to compute thus making no more than $O(1)$ calls to the primitives below them. The number of calls to a primitive in a leaf of the tree is equal to $c^{\log n}$ where c is a constant. With some simple math we calculate the number of calls to be equal to n. This is only true if the order of the base of the log is the same as the order of the base of the exponentiation. Thus the total running time is polynomial in n and in m.

To make a nice statement about this we will distinguish general combiners and very efficient ones.

Definition 12.13 *A Combiner is said to be very efficient if its running time is bounded by a constant times the running time of its candidates.*

Examples of very efficient combiners are the ones for OWF, pseudo random generators and KA(when functionality is assumed).

Lemma 12.14 *To build a $(1,n)$-Combiner from $(1,2)$-Combiners the $(1,2)$-Combiners have to be very efficient combiners.*

12.6 Universal schemes for Primitives

A nice theoretical purpose of Robust Combiners is the creation of schemes which are an implementation of a certain primitive if such an implementation exists. These constructions are called universal schemes.

Definition 12.15 (Universal Schemes) *A universal scheme $\mathcal{U}$ for a cryptographic primitive $\mathcal{P}$ is an explicit construction with the property that if the primitive $\mathcal{P}$ exists, then $\mathcal{U}$ is a secure implementation of $\mathcal{P}$.*

Lemma 12.16 *For any cryptographic primitive $\mathcal{P}$, a Universal-$\mathcal{P}$ scheme can be provided if:*

1. *There is a known polynomial $p(\cdot)$ such that if there exists an implementation for $\mathcal{P}$ then there also exists an implementation for $\mathcal{P}$ with running time bounded by $p(n)$.*
2. *$\mathcal{P}$ admits $(1, k)$-robust combiners for k a super-constant ($\omega(1)$) in the security parameter n.*

The idea of an universal scheme $\mathcal{U}$ is to include every possible implementation of a primitive $\mathcal{P}$. If the security parameter is set to n, all Turing machines of description length at most $\log n$ are allowed to participate in the construction of the combiner. The reason for only allowing programs of description length $\log n$ is that the number of possible programs is $O(n)$. If it would be larger running time and storage space would be exponential. The $(1, n)$-combiner has to run in polynomial time of course. If a program exists for large enough n that correctly implements our primitive $\mathcal{P}$ this program is included in $\mathcal{U}$. By the proven robustness of the combiner we have that $\mathcal{U}$ is also an implementation of $\mathcal{P}$.

Of course we still have to meet the first requirement of lemma 12.16. This can be done making use of the so called padding argument. A padding argument is a method for transferring results about one complexity bound to another complexity bound, by padding extra dummy characters onto the inputs of the machines involved. If we choose a number of steps l a primitive is allowed to execute then by running the primitive with increasing security parameter we can find a parameter s such that the running time will be $\leq l$ in l^2. Now if we execute the primitive with security parameter s it will have a running time $\leq l$.

12.7 Summary

In this chapter we have seen quite an introduction to robust combiners/tolerant constructions. We have seen what they are, why they are nice, how they work and what you can do with them. Lots of examples make this a very simple and readable chapter, I think. We have seen combiners for encryption, one way functions, commitment, signature schemes, key agreement and hash functions. Also we have seen some general approaches of combiners and some theoretic applications. It would be nice if people would use these simple elegant constructions more in the future.

I also hope this chapter will encourage people to read more about these constructions especially [HKN+05] and [Her05]. These papers are a lot harder to read, especially the first one, which introduces a $(2, 3)$-Robust Combiner for Oblivious Transfer. The second paper is introducing the constructions for commitment we have seen in this chapter. A part of that paper I did not cover is about arbitrary compositions of constructions.

Bibliography

[AB81] Asmuth, C. A. and Blakley, G. R. An efficient algorithm for constructing a cryptosystem which is harder to break than two other cryptosystems. *j-COMP-MATH-APPL* **7**, 6 (1981), 447–450.

[Arm88] Armstrong, M. A. *Groups and symmetry.* Springer-Verlag GmbH, 1988.

[BR93] Bellare, M. and Rogaway, P. Random oracles are practical: A paradigm for designing efficient protocols. In proc. *ACM Conference on Computer and Communications Security* (1993), pp. 62–73.

[CDS94] Cramer, R., Damgård, I., and Schoenmaker, B. Proofs of partial knowledge and simplified design of witness hiding protocols. *Lecture Notes in Computer Science, Springer-Verlag* **839** (1994), 174–187.

[CGH04] Canetti, R., Goldreich, O., and Halevi, S. The random oracle methodology, revisited. *J. ACM* **51**, 4 (2004), 557–594.

[Cia80] Cialdini, R. *Influence: Science and practice, (2nd Ed.).* Scott, Foresman, & Company, 1980.

[CSI05] CSI/FBI. Computer crime and security survey.

[CT05] Chew, M. and Tygar, J. D. Collaborative filtering captchas. In proc. *HIP* (2005), pp. 66–81.

[Deh04] Dehornoy, P. Braid-based cryptography. *Contemporary Mathematics* **360** (2004).

[ECH$^+$92] Epstein, D., Cannon, J., Holt, D., Levy, S., dn, M. P., and Thurston, W. *Word Processing in Groups.* Jones and Bartlett, Boston, MA, 1992.

[FGMY97] Frankel, Y., Gemmell, P., MacKenzie, P. D., and Yung, M. Optimal resilience proactive public-key cryptosystems. In proc. *IEEE Symposium on Foundations of Computer Science* (1997), pp. 384–393.

[FMM+02] Furukawa, J., Miyauchi, H., Mori, K., Obana, S., and Sako, K. An implementation of a universally verifiable electronic voting scheme based on shuffling. In proc. *Financial cryptography 2002* (Berlin, 2002), Springer-Verlag, pp. 16–30.

[FOO93] Fujioka, A., Okamoto, T., and Ohta, K. A practical secret voting scheme for large scale elections. In proc. *ASIACRYPT '92: Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques* (London, UK, 1993), Springer-Verlag, pp. 244–251.

[FS90] Feige, U. and Shamir, A. Witness indistinguishable and witness hiding protocols. In proc. *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing* (New York, NY, USA, 1990), ACM Press, pp. 416–426.

[GJKR96] Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. Robust and efficient sharing of rsa functions. In proc. *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology* (London, UK, 1996), Springer-Verlag, pp. 157–172.

[GM84] Goldwasser, S. and Micali, S. Probabilistic encryption. *J. Comput. Syst. Sci.* **28**, 2 (1984), 270–299.

[GNR+01] Gerck, E., Neff, C. A., Rivest, R. L., Rubin, A. D., and Yung, M. The business of electronic voting. In proc. *Financial cryptography 2001* (Berlin, 2001), Springer-Verlag, pp. 243–268.

[Her05] Herzberg, A. On tolerant cryptographic constructions. In proc. *CT-RSA* (2005), A. Menezes (ed.), vol. 3376 of *Lecture Notes in Computer Science*, Springer, pp. 172–190.

[HG98] Hartung, F. and Girod, B. Watermarking of uncompressed and compressed video. *Signal Processing* **66** (1998), 283–301.

[HKN+05] Harnik, D., Kilian, J., Naor, M., Reingold, O., and Rosen, A. On robust combiners for oblivious transfer and other primitives. In proc. *EUROCRYPT* (2005), R. Cramer (ed.), vol. 3494 of *Lecture Notes in Computer Science*, Springer, pp. 96–113.

[HR00] Honsinger, C. and Rabbani, M. Data embedding using phase dispersion, 2000. http://www.kodak.com/US/plugins/acrobat/en/corp/ researchDevelopment/dataEmbedding.pdf.

[IAG99] Iris Anshel, M. A. and Goldfeld, D. An algebraic method for public-key cryptography. *Mathematical Research Letters* **6** (1999), 287–291.

[Jou04] Joux, A. Multicollisions in iterated hash functions. application to cascaded constructions. In proc. *CRYPTO* (2004), M. K. Franklin (ed.), vol. 3152 of *Lecture Notes in Computer Science*, Springer, pp. 306–316.

[KH02] KANKANHALLI, M. S. AND HAU, K. Watermarking of electronic text documents. *Electronic Commerce Research* **1** (2002), 169–187.

[KLC$^+$00] KO, K. H., LEE, S. J., CHEON, J. H., HAN, J. W., KANG, J. SUNG, AND PARK, C. New public-key cryptosystem using braid groups. In proc. *Advances in Cryptology - CRYPTO 2000* (2000), S.-V. GmbH (ed.), vol. 1880, pp. 166–184.

[KM05] KWON, T. AND MOON, H. Multi-modal biometrics with pki technologies for border control applications. In proc. *Lecture Notes in Computer Science* (2005).

[Kra03] KRAWCZYK, H. Sigma: The 'sign-and-mac' approach to authenticated diffie-hellman and its use in the ike-protocols., 2003.

[KY02] KATZ, J. AND YUNG, M. Threshold cryptosystems based on factoring. In proc. *ASIACRYPT '02: Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security* (London, UK, 2002), Springer-Verlag, pp. 192–205.

[Len00] LENSTRA, A. K. Integer factoring. *Designs, Codes and Cryptography* **19** (2000), 101–128.

[MNPS04] MALKHI, D., NISAN, N., PINKAS, B., AND SELLA, Y. Fairplay — a secure two-party computation system, 2004.
`http://www.usenix.org/events/sec04/tech/malkhi/malkhi.pdf`.

[MP01] MALKHI, D. AND PAVLOV, E. Anonymity without 'cryptography', 2001.

[MS02] MITNICK, K. AND SIMON, W. *The Art of Deception, Controlling the Human Element of Security.* Wiley Publishing, Inc, 2002.

[NST00] NSTISSC. National information systems security (infosec) glossary.

[PLB$^+$05] PETITCOLAS, F., LEE, J.-H., BRUNET, M., KATZENBEISSER, S., KUTTER, M., ANKAPURA, M., HANSEN, K., ET AL. Digital watermarking frequently asked questions (faq), 2005.
`http://www.watermarkingworld.org/faq.html`.

[Pol74] POLLARD, J. Theorems on factorization and primality testing. *Proc. Cambr. Philos. Soc.* **76** (1974), 521–528.

[Pol78] POLLARD, J. Monte-carlo methods for index computation (mod p). *Mathematics of Computation* **32(143)** (1978), 918–924.

[PSM82] PICKHOLTZ, SCHILLING, D., AND MILSTEIN, L. Theory of spread-spectrum communications - a tutorial. *IEEE Trans. Comm.* **30** (1982), 855–884.

[Rab98] RABIN, T. A simplified approach to threshold and proactive rsa. In proc. *CRYPTO* (1998), pp. 89–104.

[RBO89] RABIN, T. AND BEN-OR, M. Verifiable secret sharing and multiparty protocols with honest majority. In proc. *STOC '89: Proceedings of the twenty-first annual ACM symposium on Theory of computing* (New York, NY, USA, 1989), ACM Press, pp. 73–85.

[RPFJ04] RYGER, R., PINKAS, B., FEIGENBAUM, J., AND JEAN, F. S. Secure computations of surveys, 2004.
`http://www.cs.yale.edu/homes/jf/SMP2004.pdf`.

[S74] S, M. *Obedience to authority.* Harper and Row, 1974.

[SCP98] SANTIS, A. D., CRESCENZO, G. D., AND PERSIANO, G. Communication-efficient anonymous group identification. In proc. *CCS '98: Proceedings of the 5th ACM conference on Computer and communications security* (New York, NY, USA, 1998), ACM Press, pp. 73–82.

[Sha79] SHAMIR, A. How to share a secret. *Commun. ACM* **22**, 11 (1979), 612–613.

[SHG98] SU, J. K., HARTUNG, F., AND GIROD, B. Digital watermarking of text, image and video documents. *Comput. & Graphics* **22**, 6 (1998).

[Sho97] SHOUP, V. Lower bounds for discrete logarithms and related problems. *Lecture Notes in Computer Science* **1233** (1997), 256–266.

[Shp04] SHPILRAIN, V. Assessing security of some group based cryptosystems. *Contemporary Mathematics* **360** (2004).

[Sou02] SOUTAR, C. Biometric system performance and security. On line availible at `http://www.bioscrypt.com/assets/bio_paper.pdf`, 2002.

[SRs+99] SOUTAR, C., RODBERGE, D., STIOANOV ALEX, GILROY, R., AND KUMAR, B. V. Biometric encryption. On line availible at `http://www.bioscrypt.com/assets/Biometric_Encryption.pdf`, 1999.

[Stu02] STUDHOLME, C. The discrete log. http://www.cs.toronto.edu/ cvs/dlog/research_paper.ps, 2002.

[Tel02] TEL, G. *Cryptografie.* Addison-Wesley, 2002.

[TP91] TURK, M. AND PENTLAND, A. Eigenfaces for recognition. *Journal of cognitive neuroscience* **3**, 1 (1991), 71–86.

[Yao86] YAO, A. How to generate and exchange secrets, 1986.

[ZHBJ73] ZIMBARDO, P. G., HANEY, C., BANKS, W. C., AND JAFFE, D. The mind is a formidable jailer: A pirandellian prison. *The New York Times Magazine, Section 6, pp. 38, ff* (1973).

Index