Commitment Schemes and Zero-Knowledge Protocols (2008)

Ivan Damgård and Jesper Buus Nielsen
Aarhus University, BRICS

Abstract

This article is an introduction to two fundamental primitives in cryptographic protocol theory: commitment schemes and zero-knowledge protocols, and a survey of some new and old results on their existence and the connection between them.

1 What's in this article?

This article contains an introduction to two fundamental primitives in cryptographic protocol theory: commitment schemes and zero-knowledge protocols. We also survey some new and old results on their existence and the connection between them. Finally, some open research problems are pointed out.

Each of the two main sections (on commitments, resp. zero-knowledge) contain its own introduction. These can be read independently of each other. But you are well advised to study the technical sections on commitment schemes before going beyond the introduction to zero-knowledge.

The reader is assumed to have some basic knowledge about cryptography and mathematics, in particular public key cryptography and the algebra underlying the RSA and discrete log based public key systems.

2 Commitment Schemes

2.1 Introduction

The notion of *commitment* is at the heart of almost any construction of modern cryptographic protocols. In this context, making a commitment simply means that a player in a protocol is able to choose a value from some

(finite) set and commit to his choice such that he can no longer change his mind. He does not however, have to reveal his choice - although he may choose to do so at some later time.

As an informal example, consider the following game between two players P and V:

1. P wants to commit to a bit b. To do so, he writes down b on a piece of paper, puts it in a box, and locks it using a padlock.
2. P gives the box to V
3. If P wants to, he can later *open* the commitment by giving V the key to the padlock.

There are two basic properties of this game, which are essential to any commitment scheme:

- Having given away the box, P cannot anymore change what is inside. Hence, when the box is opened, we know that what is revealed really was the choice that P committed to originally. This is usually called the *binding property*.
- When V receives the box, he cannot tell what is inside before P decides to give him the key. This is usually called the *hiding property*

There are many ways of realizing this basic functionality, some are based on physical processes, e.g. noisy channels or quantum mechanics, while others are based on distributing information between many players connected by a network. We will say a bit more about this later, but for now we will concentrate on the scenario that seems to be the most obvious one for computer communication: commitments that can be realized using digital communication between two players.

As a very simple example of this kind of commitments, consider the case where P has a pair of RSA keys, where V (like anyone else) knows the public key with modulus n and public exponent e. To commit to a bit b, P can build a number x_b, which is randomly chosen modulo n, such that its least significant bit is b. Then he sends the encryption $C = x_b^e \bmod n$ to V. We do not prove anything formal about this scheme here, although that is in fact possible. But it should be intuitively clear that P is stuck with his choice of b since the encryption C determines all of x_b uniquely, and that V will have a hard time figuring out what b is, if he cannot break RSA. Thus, at least intuitively, the binding and hiding requirements are satisfied.

Why should we be interested in building such commitment schemes? Primarily because this simple functionality enables the construction of secure protocols that accomplish surprisingly complicated, even seemingly impossible tasks. We will see some examples of this in the section on zero-knowledge. But we can already now give an example of a simple problem that seems intractable without commitment schemes, namely *coin flipping by telephone*.

The following story was introduced by Manuel Blum: suppose our old friends Alice and Bob are getting a divorce. They are at the point where they cannot even stand facing each other, so they have to discuss over the phone how to split the furniture, the kids, etc. But one problem remains: *who gets the car?* Since they cannot agree, they decide to flip a coin. However, they quickly realize that this is easier said than done in this situation where of course they don't trust each other at all. Bob would not be very happy about a protocol where he announces $HEADS$, only to hear Alice reply on the phone: "Here goes...I'm flipping the coin....You lost!". How can we solve this? Well, certainly not by asking Alice to flip the coin and announce the result to Bob before he has chosen heads or tails; Alice would be just as unhappy as Bob was before. We seem to be in a deadlock - neither party wants to go first in announcing their choice. However, this deadlock can be broken: using a commitment scheme, we get the following simple protocol:

1. Alice commits to a random bit b_A, and sends the resulting commitment C to Bob (you can think of C as being a locked box or an encryption, as you wish).
2. Bob chooses a random bit b_B and sends it to Alice.
3. Alice opens C to let Bob learn b_A, and both parties compute the result, which is $b = b_A \oplus b_B$.

It is not hard to argue intuitively that if the commitment is binding and hiding, then if at least one of Alice and Bob play honestly and chooses a bit randomly, then the result is random, no matter how the other party plays. A formal proof requires a more precise definition of commitments, which we will get to in the next section.

2.2 (In)distinguishability of Probability Distributions

Before we can define formally the security of commitment schemes, we need to define some auxiliary concepts.

The first concept is: what does it mean for an entity, typically a probability, to be negligible – "too small to matter"? There are different ways

in which one can define what negligible means, from the point of view of a practical application, one might want to say that anything occurring with probability below a concrete bound, such as 2^{-50}, is negligible. In complexity based cryptography, one usually prefers an asymptotic definition:

Definition 1 $\epsilon(l)$ is *negligible in* l if for any polynomial p, $\epsilon(l) \leq 1/p(l)$ for all large enough l. □

One motivation for this is that if we perform repeatedly an experiment in which a particular event occurs with negligible probability $\epsilon(l)$, then the expected number of repetitions before seeing an occurrence is $1/\epsilon(l)$ which is super-polynomial in l. In this sense we can say that events that occur with negligible probability occur so seldom that polynomial time algorithms will never see them happening.

The next concept is that of Indistinguishability. Consider first a probabilistic algorithm U. If we run U on input string x, we cannot be sure that one particular string is output since U makes random choices underway. What we have is instead a probability distribution: for every possible string y there some probability that y is output when x was the input. We will call this probability $U_x(y)$, and U_x will stand for the probability distribution of U's output, on input x. Note that we do not require U to be polynomial time, but we will require throughout that the length of U's output is polynomial in the input length.

Next, consider two probabilistic algorithms U, V. Say we do the following experiment: we run both U and V on the same input x, and we choose one of the outputs produced, call it y. We then give x and y to a third party D which we call the *distinguisher*, and ask him to guess which of the two algorithms we used to produce y. Informally, we say that U, V are indistinguishable if D has, in some sense, a hard time.

It is not hard to see why such a concept is relevant in cryptography: suppose the input x is a public encryption key, and U encrypts a message m_1 while V encrypts a message m_2. Now, indistinguishability of U and V exactly means that it is difficult to guess whether a ciphertext contains m_1 or m_2, and this is the same as standard definition of security for public-key systems, known as semantic security.

It should be clear from the above that all that matters when we ask whether U, V are indistinguishable, are the output distributions U_x, V_x generated for each input. Therefore, whether we think of U, V as probabilistic algorithms or as families of probability distributions makes no difference, and we will use both terminologies in the following. However, to use the

indistinguishability concept we have to be more precise, in particular, we need:

Definition 2 Given two probability distributions P, Q, the *statistical distance* between them is defined to be $SD(P,Q) = \sum_y |P(y) - Q(y)|$, where $P(y)$ (or $Q(y)$) is the probability P (or Q) assigns to y. □

Using this, we have

Definition 3 Given two probabilistic algorithms (or families of distributions) U, V, we say that

- U, V are perfectly indistinguishable, written $U \sim^p V$, if $U_x = V_x$ for every x.
- U, V are statistically indistinguishable, written $U \sim^s V$, if $SD(U_x, V_x)$ is negligible in the length of the string x.
- U, V are computationally indistinguishable, written $U \sim^c V$, if the following holds for every probabilistic polynomial time algorithm D: let $p_{U,D}(x)$ be the probability that D outputs "U" as its guess, when D's input comes from U, and similarly $p_{V,D}(x)$ be the probability that D outputs "U" as its guess, when D's input comes from V. Then $|p_{U,D}(x) - p_{V,D}(x)|$ is negligible in the length of x. An alternative equivalent definition: Let $D(U)$ be the algorithm that first runs U on input x to get result y, and then runs D on input y and x. Define $D(V)$ similarly. Then $U \sim^c V$ if and only if $D(U) \sim^s D(V)$ for every probabilistic polynomial time algorithm D [1].

Sometimes we do not want to consider how U, V behave on arbitrary input x, but only when x is in some language L. We then say, e.g. $U \sim^c V$ on input $x \in L$. □

Some remarks on the meaning of this definition: if $U \sim^p V$, D has no chance of distinguishing at all, no matter how much computing power it has. It might as well flip a coin to decide who produced the result it is given. If $U \sim^s V$, D may have a small advantage over a random guess, but it will remain negligible, no matter how much computing power D has. Finally, if $U \sim^c V$, the distributions U_x, V_x may be totally different, but it requires a lot of computing power to tell the difference, so D's chances of success remain small if it is limited to polynomial time.

[1] In the standard definition from the literature D is allowed to be a *non-uniform algorithm*, i.e., specified by a polynomial size family of circuits. However, this makes no difference for the purpose of this note

2.3 Defining Commitment Schemes

We now come back to commitment schemes. Two things are essential in the RSA example:

- The RSA key used does not come falling from the sky. There has to be an algorithm for generating it: some procedure that takes as input the length of modulus to generate, and then chooses randomly n and e, suitable for use as an RSA public key. In the example this algorithm would be run by P initially, and P must have some confidence that keys generated by this algorithm cannot be easily broken by V.
- When committing, it is essential that P makes random choices. The scheme above (in fact any scheme) would be completely insecure, if this was not the case (see Exercise 4). Thus the commitment sent to V must be a function of both the bit committed to, and of some random choices made by P.

Keeping this in mind, we can abstract the following general definition. It is somewhat simplified in that it does not cover all commitment schemes, but it covers the examples we will look at, and is enough to get a feeling for how such definitions work.

We will think of a commitment scheme as being defined by a a probabilistic polynomial time algorithm $\mathcal{G}$ called a *generator*. It takes as input 1^l where l is a security parameter and corresponds to e.g. the length of RSA modulus we want. It outputs a string pk, the *public key* of the commitment scheme. The scheme defines for every public key pk a function $\texttt{commit}_{pk} : \{0,1\}^l \times \{0,1\} \to \{0,1\}^l$. where the idea is that a 0/1-values can be committed to. We stick to *bit*-commitments here for simplicity [2].

To use the scheme in practice, one first executes a *set-up phase* (once and for all) where either P or V runs $\mathcal{G}$, and sends the public key pk to the other party. In some schemes it is necessary in addition to convince the other party that pk was correctly chosen, in case this is not easy to verify directly. Thus, one of the parties may *reject* in the set-up phase, meaning that it refuses to use the public key it received.

Assuming that the public key was accepted, to commit to a bit b, P chooses r at random from $\{0,1\}^l$ and computes the commitment $C \leftarrow \texttt{commit}_{pk}(r,b)$. To open a commitment, r, b are revealed, and V checks that indeed $C = \texttt{commit}_{pk}(r,b)$.

[2] Also for simplicity, this description insists that for security parameter l, one uses l bits of random input to commit, and commitments will have length l bits. In general, one can allow any polynomial in l here without problems.

To define precisely the two essential properties of hiding and binding for this kind of commitment, we have to distinguish two flavors of commitment schemes:

Unconditional binding and Computational hiding For this type of scheme, P will run the key generator and send the public key to V, who verifies the key and either accepts or rejects.

***unconditional binding*:** Means that even with infinite computing power, P cannot change his mind after committing. We require that if pk is correctly generated, then b is uniquely determined from $\texttt{commit}_{pk}(r, b)$, i.e., for any c, there exists at most one pair (r, b) such that $c = \texttt{commit}_{pk}(r, b)$. Furthermore, an honest V accepts an incorrectly generated pk with probability negligible in l.

***computational hiding*:** Means that a polynomially bounded V will have a hard time guessing what is inside a commitment. We require that $(pk, \texttt{commit}_{pk}(r, 0)) \sim^c (pk, \texttt{commit}_{pk}(s, 1))$.

Computational binding and Unconditional Hiding For this type of scheme, V will run the key generator and send the public key to P, who verifies the key and either accepts or rejects.

***computational binding*:** Means that unless you have "very large" computing resources, then you chances of being able to change your mind are very small. More precisely: take any probabilistic polynomial time algorithm P^* which takes as input a public key produced by the generator $\mathcal{G}$ on input 1^l. Let $\epsilon(l)$ be the probability (over the random choices of $\mathcal{G}$ and P^*) with which the algorithm outputs a commitment and two valid openings revealing distinct values. That is, it outputs C, b, r, b', r' such that $b \neq b'$ and $\texttt{commit}_{pk}(r, b) = C = \texttt{commit}_{pk}(r', b')$. Then $\epsilon(l)$ is *negligible in* l.

***unconditional hiding*:** Means that a commitment to b reveals (almost) no information about b, even to an infinitely powerful V. We require that if we restrict to correctly generated pk's, then $\texttt{commit}_{pk}(r, 0) \sim^s \texttt{commit}_{pk}(s, 1)$ (for random independent r, s). Furthermore an honest P should accept an incorrectly generated pk with at most negligible probability. In the best possible case, we have $\texttt{commit}_{pk}(r, 0) \sim^p \texttt{commit}_{pk}(s, 1)$ and P never accepts

a bad pk, i.e. commitments reveal no information whatsoever about the committed values. We then speak of *perfectly* hiding commitments.

Before we continue, a word of warning about the definitions of the computational flavours of hiding and binding: They are based on the asymptotic behaviour of an adversary as we increase the value of the security parameter. This is mathematically convenient when doing proofs, and has nice connections to standard complexity theory - but one should take care when evaluating the meaning in practice of results according to such a definition: it implicitly classifies everything that can be solved in probabilistic polynomial time as being "easy" and anything else as being "hard", and this distinction is not always accurate in real life. Even if a problem (such as breaking a commitment scheme) is asymptotically hard, it might still be easy in practice for those input sizes we want in a particular application. This does not at all mean that asymptotic security results are worthless, only that usage of a scheme in real life should always be supplemented with an analysis of practical state of the art of solutions to the (supposedly) hard problem we base ourselves on.

It is evident that an unconditional guarantee is better than a computational one, so why don't we try to build commitments that are both unconditionally binding *and* hiding? Well, unfortunately this is impossible!

Imagine we had such a scheme. Then, when P sends a commitment to e.g. 0 $C = \texttt{commit}_{pk}(r, 0)$, there must exist an r', such that $C = \texttt{commit}_{pk}(r', 1)$. If not, V could conclude that the committed value could not be 1, violating unconditional hiding. He could come to this conclusion by simply trying all possibilities for r'. This may a long time, but this is irrelevant when we talk about unconditional security. But then, if P has unlimited computing power, he can find r' and change his mind from 0 to 1, violating unconditional binding. This reasoning does not depend on the particular definition we have presented of commitment schemes. It extends to any protocol whatsoever for committing to a value in a two-player game. The basic reason for this is that the scenario by definition ensures that each player sees everything the other player sends.

There are several scenarios, however, where this reasoning does not apply. In a distributed scenario with many players, or in a two-party case where communication is noisy, it is no longer true that V sees exactly what P sends and vice versa. And in such cases, unconditional binding and hiding can in fact be obtained simultaneously. For commitment schemes in such scenarios, see e.g. [10, 3, 14, 9]. Note, however, that despite the fact

that the reasoning does not apply to quantum communication either, bit commitment with unconditional security is not possible with quantum communication alone. [3]

Exercise 1 [negligible] Call a function $f : \mathbf{N} \to \mathbf{R}$ *polynomial in* l if there exist polynomial p and constant l_0 such that $f(l) \leq p(l)$ for all $l > l_0$. Recall that a function $\epsilon : \mathbf{N} \to \mathbf{R}$ *negligible in* l if for all polynomials p there exists a constant l_p such that $\epsilon(l) \leq 1/p(l)$ for all $l > l_c$.

1. Prove that if ϵ and δ are negligible in l, then $\epsilon + \delta$ is negligible in l.

2. Prove that if ϵ is negligible in l and f is polynomial in l, then $f \cdot \epsilon$ is negligible in l.

Exercise 2 [statistical distance] Consider algorithms U, V, W. Show that if $U \sim^s V$ and $V \sim^s W$, then $U \sim^s W$.

Exercise 3 Show that if $U \sim^c V$ and $V \sim^c W$, then $U \sim^c W$. Hint: use the previous exercise and the alternative definition of $\sim^c$.

Exercise 4 [commitments must be randomised] This exercise concerns the necessity to make commitment schemes randomised (in the sense that the output of $\texttt{commit}_{pk}(r, m)$ depends on r and not just m).

1. Prove that a commitment scheme which is perfectly binding, but does not depend on the random input r, cannot be hiding in any flavor.

2. Say that a commitment scheme is g-randomised if it only depends on the g first bits of r. If a commitment scheme is perfectly binding and$\log l$-randomized, can it be computationally hiding? What if we replace $\log l$ by $c \log l$ for a constant c? or by $(\log l)^c$?

2.4 Examples of Commitment Schemes

Many examples of commitment schemes have been suggested in the literature, see e.g. [4] for some basic ones or [11] for some later and more efficient examples.

Above, we have seen an example based on RSA with unconditional binding. This scheme also satisfies computational hiding, assuming that the RSA

[3] By quantum communication, we mean a scenario where information is sent encoded in the state of very small physical systems, such as single elementary particles.

encryption function is hard to invert, although this is quite technically complicated to prove. It does *not* follow immediately, since a priori it might well be the case that the least significant bit of x is easy to compute from $x^e \bmod n$, even though all of x is hard to find. However in [1] it was proved that this is not the case: any algorithm that guesses the least significant bit of x with probability slightly better than $1/2$ can, by a randomised polynomial time reduction, be turned into one that inverts the RSA encryption function.

Of course, any secure public key encryption scheme, where validity of the public key can be checked efficiently, can also be used as a commitment scheme.

We now look at a general way to make commitment schemes with unconditional hiding. We consider first a construction based on RSA as an example Consider any algorithm for generating on input 1^l a secure RSA l-bit modulus n. We can extend this by choosing also a prime $q > n$, and finally defining $f(x) = x^q \bmod n$. Assuming that RSA with modulus n and public exponent q is secure, f is a one-way function, i.e., given $f(x)$, it is hard to compute x. Moreover, f is a *homomorphism*, i.e., $f(1) = 1$ and $f(xy) = f(x)f(y)$, Finally q, being a prime larger than n, must be prime to $\phi(n)$ and so q is a valid RSA exponent which means that one can directly check from n, q that f is a 1-1 mapping on Z_n^*. The algorithm for selecting n, q will be called $\mathcal{H}$, and our basic assumption is the following:

RSA assumption: Suppose we run $\mathcal{H}$ on input 1^l to get n, q, choose x at random in Z_n^*, and run any probabilistic polynomial time algorithm A on input $n, q, f(x)$. Then the probability that A outputs x is negligible in l.

We now build an unconditionally hiding commitment scheme as follows.

- **Set-up Phase**: the key generator $\mathcal{G}$ for the commitment scheme is defined based on $\mathcal{H}$ as follows: it runs $\mathcal{H}$ on input 1^l. It then chooses a random element $x \in Z_n^*$ and outputs as public key n, q and $y = f(x) = x^q \bmod n$. In the set-up phase, V runs $\mathcal{G}$ and sends the output n, q, y to P, who checks that $y \in Im(f) = Z_n^*$ (i.e. check that $gcd(y, n) = 1$).

- **Commit function**: is defined as a mapping from $Z_n^* \times \{0, 1\}$ to G. Concretely, $\texttt{commit}_{pk}(r, b) = y^b f(r) \bmod n$.

- **Hiding Property**: is unconditionally satisfied, since P can verify without error that q is a valid exponent and that $y \in Im(f)$, and in this case a commitment to b will have distribution independent of b, namely the uniform distribution over Z_n^*. This is because P chooses

r uniformly in Z_n^*, f is a 1-1 mapping and hence $f(x)$ is also uniform in Z_n^*. Finally, multiplication by the constant y is also a one-to-one mapping in the group Z_n^*, so $yf(x) \bmod n$ is uniform as well. Thus these commitments are in fact perfectly hiding.

- **Binding Property**: Follows from the following fact: say we are given an algorithm A that breaks the binding property of this scheme with success probability ϵ in time T_A. Then there exists an algorithm A' that breaks RSA encryption as generated by $\mathcal{H}$ with success probability ϵ as well and in time T_A plus the time needed for one inversion and one multiplication in Z_n^*. Thus existence of such an A implies existence of an algorithm A' that contradicts our assumption on $\mathcal{H}$.

  This is easy to show: say we are given n, q and want to find the plaintext for ciphertext $y \in Z_n^*$. We run A on input n, q, y pretending this is the public key of a commitment scheme instance. A outputs in time T_A a commitment c and openings $r_0, 0$ and $r_1, 1$. We now output $x = r_0 r_1^{-1} \bmod n$. We leave it to the reader to show that if indeed $r_0, 0$ and $r_1, 1$ are valid openings of c, then $f(x) = y$.

There are several things to notice about this scheme and its proof:

- Actually, the only essential thing we need from our function f is the fact that it is a one-way function and a homomorphism. Therefore, the idea of the construction is not limited to being based on RSA. There are many other possibilities. For instance, we can also base ourselves on the discrete log problem. In this case, f would be of the form $f(x) = g^x \bmod p$ for a large prime p (see Exercise 6).

- In the set-up phase, it is essential that P is convinced that $y \in Im(f)$. It would be devastating if V could get away with selecting $y \notin Im(f)$, which might be the case if f is not surjective (see Exercise 5). For our RSA example, this was not a problem: P can check for himself that f is surjective which implies that $y \in Im(f)$. In other cases, the set-up phase must be more elaborate in that V must convince P that the public key was correctly selected. This can be done using a zero-knowledge protocol (see Section 3). In particular it is always possible given any homomorphism f for V to convince P in zero-knowledge that $y \in Im(f)$, which is in fact enough for the scheme to be secure.

- The proof of the binding property is an example of so called proof by *black-box reduction*: we want to show that existence of cryptographic

primitive $P1$ implies existence of primitive $P2$. In our case $P1$ = secure RSA encryption and $P2$ = unconditionally binding commitments schemes.

To do this, we first make a construction that takes an instance of $P1$ and builds an instance of $P2$. We then show that any algorithm that can break $P2$ can be used to build an algorithm that breaks $P1$ "just as efficiently". This is done by a reduction that treats the algorithm attacking $P2$ as a black-box: it doesn't care how the algorithm manages to break $P2$, it just uses the fact that it succeeds in doing so. We conclude that if the security properties of $P2$ are violated, so are those of $P1$, and conversely, if secure instances of $P1$ exist so do secure instances of $P2$.

This black-box paradigm has proven extremely productive in many areas of cryptography.

- The black-box reduction we built to show the binding property is actually much stronger than needed for the definitions: for that, it would have been enough if we had shown that the running time of A' was polynomial in T_A, and that the success probability of A' was a polynomial function of ϵ. Still, what we have done is far from being overkill: what we want, in practice as well as in theory, is basically to say that "breaking the commitment scheme is just as hard as it is to invert the homomorphism". And of course we can make this claim in a stronger sense, the more efficient a reduction we have. Hence if we want results that are meaningful not only in theory, but also in practice, it is important to try to obtain as efficient a reduction as possible in any proof of this type.

- Group homomorphisms can also be used to build unconditionally binding commitments, and to build schemes where one can commit to many bits in the same commitment. For details on this, see [11].

Exercise 5 [group-homomorphism commitments] This exercise deals with the the unconditionally hiding commitment scheme based on RSA.

1. Finish the proof of the binding property.

2. Here you will show that we have to force V to choose q as a valid RSA exponent. For instance, $q = 2$ would be a disaster. In this case $f(x) = x^2 \bmod n$ is a 4 to 1 mapping, i.e., every element in $Im(f)$ has 4 preimages (you may assume this without proof).

Show that if $y \notin Im(f)$ then $yf(r) \bmod n \notin Im(f)$ for any r. Furthermore it is a known fact (which you may assume) that if you are given the factorization of n, it is easy to tell if a given number is in $Im(f)$ or not. Use this to show that if $q = 2$ is allowed, then V can generate a public key for which he can easily compute b from $\texttt{commit}_{pk}(r, b)$.

Exercise 6 [discrete logarithm commitments] Recall that for a prime p, a generator g of the multiplicative group $\mathbf{Z}_p^*$ is an element with the property that any $a \in \mathbf{Z}_p^*$ can be written as a power of g, $a = g^i \bmod p$ doe some i.

Note that the function $f : \mathbf{Z}_{p-1} \to \mathbf{Z}_p^*, x \mapsto g^x \bmod p$ is a group homomorphism, namely $f(x + y \bmod p - 1) = f(x)f(y) \bmod p$. It is generally believed that f is one-way for a large random prime p. Use this assumption to construct an unconditionally hiding and computationally binding commitment scheme — use the RSA construction as inspiration, and write down the commitment function explicitly. Argue that if one can efficiently break the binding property, one can also efficiently solve the discrete log problem mod p, i.e., invert f.

Exercise 7 [string-commitments, group-homomorphism commitments] This exercise considers a generalisation of the unconditionally hiding bit-commitment scheme based on a one-way group homomorphism, to allow to commit to several bits.

Above we defined a bit-commitment scheme. In a *string-commitment scheme* (or *multi-bit commitment*) a public key generated by $\mathcal{G}$ on 1^l defines a function $\texttt{commit}_{pk} : \{0,1\}^o \times \{0,1\}^n \to \{0,1\}^*$, where o and n are polynomial in l and $\texttt{commit}_{pk}(r, m)$ is a commitment to $m \in \{0,1\}^n$ using randomness $r \in \{0,1\}^o$. The computational binding of a string-commitment scheme is defined as for bit-commitment scheme: for any probabilistic polynomial time algorithm A, let $\epsilon_A(l)$ be the probability that A on a random public key pk, generated by G, outputs C, m, r, m', r' such that $m, m' \in \{0,1\}^o$, $m \neq m'$, $C = \texttt{commit}_{pk}(r, m)$ and $C = \texttt{commit}_{pk}(r', m')$. Then $\epsilon_A(l)$ is negligible.

Assume now that we have a group homomorphism $f : G \to H$ where it holds that all elements of $Im(f)$ (except 1) have some known prime order q. Consider then the commitment scheme where V picks $y \in Im(f) \setminus \{1\}$ and where P commits to $m \in \{0, 1, \ldots, q-1\}$ by $\texttt{commit}_{pk}(r, m) = y^m f(r)$.[4]

1. Prove that the generalised commitment scheme is still unconditionally

[4]To fit the above definition we should actually restrict the message space to $\{0,1\}^o$ for some o such that $2^o \leq q$, consider $m \in \{0,1\}^o$ as a number $m \in \{0, 1, \ldots, 2^o - 1\}$ and then commit to m as specified.

hiding in the sense that commitments to all messages have the same distribution.

2. Prove that the generalised commitment scheme is still computationally binding. [Hint: You will need to realize and use that all non-zero numbers have a multiplicative inverse modulo the order of y.]

2.5 Theoretical Results of Existence of Commitment Schemes

It is easy to see that if any commitment scheme in the two-player model exists, then a one-way function must also exist. For example, in our definition, it is clear that the function $\texttt{commit}_{pk}$ must be one-way in order for the commitment scheme to be secure.

Hence, the optimal result is to show existence of commitment schemes based only on the existence of one-way functions. Such results are known, and for one type of commitment scheme, it follows from a result of Naor [29] (actually, Naor's result is the last in a long chain of results linking one-way functions with commitments through other primitives such a pseudorandom generators, for references on this, see [29]):

Theorem 2.1 *If one-way functions exist, then commitment schemes with unconditional binding and computational hiding exist.*

For unconditionally hiding schemes, the corresponding result is proved in [32]:

Theorem 2.2 *If one-way functions exist, then commitment schemes with unconditional hiding and computational binding exist.*

If we assume that the function we are given is not only one-way but is bijective, one can even get perfect hiding [30]. In [31], with generalisations to multi-bit commitments in [18], the following is proved:

Theorem 2.3 *If collision-intractable hash functions exist, then there exists commitment schemes with unconditional hiding and computational binding.*

Loosely speaking, a collision intractable hash function is a function $h : \{0,1\}^k \to \{0,1\}^l$ such that $l < k$, h is easy to compute, but it is hard to find $x \neq y$ such that $h(x) = h(y)$ (although such values must of course exist – for a precise definition, see [15]).

Since a collision intractable hash function is always one-way, this third result is weaker the the second, but it is still of interest from a practical

point of view: whereas the first two results involve very complex reductions and therefore lead to very inefficient commitment schemes, the third one can lead to very practical schemes.

3 Zero-Knowledge Protocols

3.1 Introduction

In order for a modern computer network to offer services related to security, it is a basic necessity that its users have access to private information, in the form of e.g. passwords, PIN codes, keys to cryptosystems, keys to signature systems etc. If I know every bit of information that you know, it will be impossible for the rest of the system to tell us apart.

This introduces a basic problem when implementing such services: of course I want my private information to stay private, but as soon as I start using it as input when computing the messages I send to other parties on the net, this introduces the risk of leaking private information, in particular if the parties I interact with do not follow the protocols, but instead do their best to maliciously trick me into revealing my secrets. This dilemma can be solved if we use protocols on the net for which we can *control exactly* how much sensitive information is being released, even in presence of adversarial behaviour. The concept of zero-knowledge, first introduced by Goldwasser, Micali and Rackoff [26], is one approach to the design of such protocols.

As an easy example, consider the classical user identification problem: we have a host computer that would like to verify the identity of users that try to log on. The classical solution is to assign a private password to each user. When logging on, the user types his user name and password, this is sent to the host, who checks it against a stored list.

The security problems with this are many and well known. Let us concentrate here on the obvious problem that if an adversary eavesdrops the line, he can pick up the password, and then impersonate the user. When trying to solve this, the immediate reaction might be to propose that we transport instead the password in a protected way. Perhaps we should just encrypt it?

But then we would be barking up the wrong tree. We have to ask ourselves first what the purpose of the protocol is. Is it to send the password from the user to the host? No! - we are trying to identify the user. What we have done initially is to assign a secret (the password) to each user, so when someone types his user name, say xxx, this is equivalent to claiming that a

certain statement is true, in this case "I know the secret corresponding to user name xxx".

The *only* thing the host needs to know here is 1 bit of information, namely whether this statement is true or not. The real purpose of the protocol is to communicate this piece of knowledge to the host. Sending the secret of the user in clear is just one way, and not even a very intelligent way to do it.

In general, we could have the user and host conduct an interactive protocol, where at the end, the host can compute a one-bit answer saying whether the user was successful in proving himself or not. Here of course we have to design the protocol such that if the user really knows the right secret, he will be successful, whereas the answer will be no, if the user is cheating and does not know the secret. If this is satisfied, we can say that the protocol really does communicate this 1 bit of knowledge saying whether the claim is true or not. But moreover, if we design the protocol correctly, we can actually obtain that it communicates *nothing more than this*. Which would mean that for example an eavesdropper listening to the communication would be just as far away from guessing the user's secret after seeing the conversation as he was before.

This leads to our first very loose definition of zero-knowledge: *a protocol is zero-knowledge if it communicates exactly the knowledge that was intended, and no (zero) extra knowledge.*

3.2 A Simple Example

One way to realize the scenario where each user has his own secret is to use a public key cryptosystem. So suppose each user A has a private key S_A known only to him, whereas everyone, including the host, knows the public key P_A.

Now, if the cryptosystem is any good, it must be the case that decrypting a ciphertext $C = P_A(M)$ is hard unless you know the private key. Hence, if you meet someone who is able to decrypt a ciphertext you send him, it is reasonable to conclude that he knows S_A, at least if you make sure that the message you encrypt is randomly chosen from a large set, such that the probability of guessing your choice is negligible. This suggests the following simple protocol, where we rename the players so that the description fits better with the definitions to follow: the user, who is the one wanting to convince the other about the truth of some claim will be called the *Prover* (P), and the host, who is interested in checking that the claim is true, will be called the *verifier* (V).

1. If the prover claims to be A, the verifier chooses a random message M, and sends the ciphertext $C = P_A(M)$ to the prover.

2. The prover decrypts C using S_A and sends the result M' to the verifier.

3. The verifier accepts the identity of the prover if and only if $M' = M$.

Let us look at this protocol from the point of view of both parties. Should the verifier be happy about this protocol? the answer is *yes* if the public key system used is secure: while the owner of S_A can always conduct the protocol successfully, an adversary who knows only the public key and a ciphertext should not be able to find the plaintext essentially better than by guessing at random.

Now what about security from the (honest) prover's point of view - is any unnecessary knowledge being communicated to the verifier here? At first sight, it may seem that everything is OK: if we consider the situation of the verifier just after sending C, then we might argue that since the verifier has just chosen the message M itself, it *already knows* what the prover will say; therefore it learns no information it didn't know before, and so the protocol is zero-knowledge.

But this reasoning is WRONG! It assumes that the verifier follows the protocol, in particular that C is generated as prescribed. This is of course unreasonable because nothing in the protocol allows the prover to check that the verifier is behaving honestly. This is more than a formal problem: assume that an adversary takes control of the verifier, and sends instead of a correctly generated C some ciphertext C' intended for the correct prover, that the adversary has eavesdropped elsewhere. And now, following the protocol, the unsuspecting prover will kindly decrypt C' for the adversary!

This is certainly not the kind of knowledge we wanted to communicate, and hence this protocol is definitely not zero-knowledge. How can we repair this protocol? The basic problem we saw is that when the verifier sends C, we are not sure if it really knows the corresponding plaintext M. If it did, we would be fine. However, the verifier will of course not be willing to reveal M immediately, since from its point of view, the purpose of the protocol is to test if the prover can compute M based only on C. And for the reasons we saw above, the prover will not be willing to go first in revealing M either. So we have a sort of deadlock situation similar to the one in the coin-flipping by telephone problem from the former section. Like that problem, this one can be solved using commitments.

Assume we have a commitment scheme that lets the prover commit to any message that can be encrypted by the public key system. Let

$\texttt{commit}_{pk}(r, M)$ denote a commitment to message M (using random choice r - we can always commit bit by bit if no more efficient methods are available). Then consider the following:

1. If the prover claims to be A, the verifier chooses a random message M, and sends the ciphertext $C = P_A(M)$ to the prover.
2. The prover decrypts C using S_A and sends a commitment to the result $\texttt{commit}_{pk}(r, M')$ to the verifier.
3. The verifier sends M to the prover.
4. The prover checks if $M = M'$. If not he stops the protocol. Otherwise he opens the commitment, i.e. he sends r, M' to the verifier.
5. The verifier accepts the identity of the prover if and only if $M' = M$ and the pair r, M' correctly opens the commitment.

Proving formally that this repair works turns out to be surprisingly complicated, but possible. The necessary techniques can be found e.g. in [5, 24]. Here, however, we are only interested in arguing informally why such a solution should have a chance of working: first, the protocol demonstrates that the prover can decrypt C based on C alone, since when the verifier finds the right plaintext inside the commitment, this shows that the prover knew it already in step 2, by the binding property of the commitment scheme. As for zero-knowledge, either the verifier knows M or not. If yes, then it can send the correct M in step 3, but then it already knows what it will find inside the commitment in step 5 and so learns nothing new. If not, then it cannot send the right value in step 3, the prover will stop the protocol, and the verifier will be left with an unopened commitment which by the hiding property is a useless piece of information that might represent any value whatsoever.

If nothing else, this example demonstrates first the fundamental role that commitments often play in protocol design, and second that we should *not* argue security of protocols based on what players *should be* doing according to the protocol, we must take any adversarial behaviour into account. Finally, it also demonstrates one basic design principle for zero-knowledge protocols that continue to appear in all sorts of incarnations: have the prover demonstrate something the verifier already knows. The problem with this is, in the above protocol as in all protocols of this type, to ensure that the verifier does indeed know in advance what the prover will say. For other examples of this kind, see e.g. the graph non-isomorphism protocol from [25].

3.3 Definitions

3.3.1 Interactive Proof Systems and Proofs of Knowledge

The protocols to follow will take place as interactions between two *Interactive Turing Machines*, i.e. ordinary probabilistic Turing machines that are in addition equipped with communication tapes allowing a machine to send and receive messages from the other one. A formal definition can be found in [26].

To define interactive proof systems, we assume that one machine, called the prover (P) has infinite computing power, and the other called the verifier (V) is polynomial time bounded. The machines get a common input string (usually called x). Running the machines on some input x results in V outputting *accept* or *reject* after which the machines halt. We say that the pair (P, V) accepts or rejects x accordingly. Finally a binary language $L \subset \{0,1\}^*$ is given.

In the previous section, we talked about the intuitive model where the prover claims that "a certain statement is true". We now specialise to the concrete case where the prover claims that a certain logical statement is true, namely that $x \in L$. This can be compared in the real world to convincing someone that a certain theorem is true. Concretely, we have the following definition [26]:

Definition 4 *The pair (P, V) is* an interactive proof system for L *if it satisfies the following two conditions:*

Completeness: *If $x \in L$, then the probability that (P, V) rejects x is negligible in the length of x.*

Soundness: *If $x \notin L$ then for any prover P^*, the probability that (P^*, V) accepts x is negligible in the length of x.*

What these conditions say is that first, the honest prover can always convince the verifier about a true statement, but that there is no strategy that convinces him about something false. Both conditions are required to hold except with negligible probability, and are in fact rather strong: even if the honest prover can convince the verifier using only polynomial computing time, there must be no way to cheat the verifier, even using infinite computing power.

There are two features that make this definition interesting, namely that interaction and error probabilities are allowed. It is easy to see that if the prover is only allowed to send a single message to the verifier, who should

then be able to check without error that the input x is in L, we would only be redefining the class NP. But with these two features, the model becomes much more powerful in terms of the class of statements that can be proved, as we shall see.

There is a variant of this, known as *Proofs of Knowledge*, where the prover's claim has a different flavour: he claims to know a certain piece of information (such as a secret key corresponding to a given public one). Such proof systems can be defined in a similar model, where however the completeness and soundness properties are replaced by *knowledge completeness* and *knowledge soundness*. The first property simply says that if the prover knows the claimed information and follows the protocol, he can almost always convince the verifier. The second, loosely speaking, says that if some prover can, using whatever strategy, convince the verifier with substantial probability, then the prover knows the information in question. By "knowing the information" we mean that the prover can compute it, and that the time he needs to do so is roughly inversely proportional to the probability with which the verifier gets convinced. A precise definition can be found in [2].

3.3.2 Interactive Arguments

Another variant of Interactive proof systems is known as *Interactive Arguments* and has perhaps more direct relations to practical protocols. In this type of protocol, we want the prover to be polynomial time, but on the other hand are only concerned about polynomial time provers cheating the verifier. This can be said to be a complexity theorist's way of modelling the situation where only realistic computing power is available to prover and verifier.

The simplest way to define an interactive argument for a language L, is to say that it is an interactive proof system, but with two changes:

- The honest prover is required to be probabilistic polynomial time, and its only advantage over the verifier is that it has a private auxiliary input. The completeness condition says that for every $x \in L$, there is an auxiliary input that allows the prover to convince the verifier almost always[5].
- The soundness condition says "for any probabilistic polynomial time prover", in stead of "for any prover".

[5]In order for the protocol to be interesting at all, the prover must have some advantage - otherwise the verifier might as well go and solve the problem on his own.

It turns out that this simplistic definition of soundness is not quite adequate in all cases, but it will do for us here. For a more complete set of definitions and a discussion of this, see [17].

3.3.3 Zero-Knowledge

Zero-Knowledge can be seen as an extra property that an interactive proof system, a proof of knowledge or an interactive argument may have. Here, we want to express the requirement that whatever strategy the verifier follows, and whatever a priori knowledge he may have, he learns nothing except for the truth of the prover's claim. We do this by requiring that assuming the prover's claim is true, the interaction between the prover and verifier can be efficiently simulated *without interacting with the prover*.

A verifier that tries to cheat the prover can be modelled by an arbitrary probabilistic polynomial time machine V^* that gets an auxiliary input δ of length at most some fixed polynomial in the length of the common input x. This represents a priori information that V^* could have e.g. from earlier executions of the protocol, which it may now use to trick the prover into revealing more information. By a conversation between P and any verifier we mean the ordered concatenation of all messages sent in an execution of the protocol. In this way, we can consider the pair (P, V^*) as a machine that gets input x and δ (only to V^*) and outputs a conversation. We now have the following [26]:

Definition 5 *An interactive proof system or argument (P, V) for language L is* zero-knowledge *if for every probabilistic polynomial time verifier V^*, there is a simulator M_{V^*} running in expected probabilistic polynomial time, such that we have $M_{V^*} \sim^c (P, V)$ on input $x \in L$ and arbitrary δ (as input to V^* only).*

For some protocols, we can obtain that $M_{V^*} \sim^p (P, V)$, or $M_{V^*} \sim^s (P, V)$ in this case we speak of *perfect zero-knowledge* respectively *statistical zero-knowledge*. Clearly, perfect zero-knowledge implies statistical zero-knowledge, which in turn implies computational zero-knowledge as defined above.

At first sight, the zero-knowledge definition may seem intuitively to contradict the proof system definition: first we say that the verifier should be convinced by talking to the prover. But then we require that the whole conversation can be efficiently simulated *without* talking to the prover – doesn't this mean that having a conversation with the prover cannot be convincing?

Fortunately, this is not the case. The explanation is that a simulator has some degrees of freedom that the prover doesn't have when executing the real protocol. In particular, the simulator can generate messages of a conversation in any order it wants - it can start with the last message first, and then try to find earlier messages that match. A real prover is forced by the verifier to proceed in the protocol with the correct time ordering of messages. And this is why it can be possible that even an infinite prover cannot cheat the verifier, and still a simulator with no special knowledge or computing power can simulate the conversation. For concreteness, see the example below.

3.4 An Example

We describe here a simple example taken from [25], namely a perfect zero-knowledge proof system for the graph isomorphism problem: the common input in this case is a pair of graphs G_0, G_1 each on n nodes, and the prover claims the graphs are isomorphic: there is a permutation π (an isomorphism) such that by permuting the nodes of G_0 according to π (and connecting two resulting nodes iff their preimages were connected in G_0), one obtains the graph G_1. We say that $\pi(G_0) = G_1$.

Note that no general probabilistic poly-time algorithm is known for deciding if two graphs are isomorphic. We will use n as a measure of the length of the input. In the protocol, we actually do not need P to be infinitely powerful, although the definition of proof systems allows this; it is enough that he knows an isomorphism π. The protocol works by repeating sequentially the following steps n times:

1. P chooses a random permutation ϕ on n points and sends $H = \phi(G_0)$ to V.
2. V chooses at random a bit b, and sends it to P.
3. If $b = 0$, P sets $\psi = \phi^{-1}$. Else he sets $\psi = \pi\phi^{-1}$. He sends ψ to V, who checks that $\psi(H) = G_b$, and rejects immediately if not.

The verifier accepts, only if all n iterations were completed successfully.

First, let us check that this is a proof system. Completeness is obvious: if indeed $\pi(G_0) = G_1$ and $\psi(G_0) = H$, then it follows trivially that V's check will be satisfied for both values of b. Soundness can be argued as follows: observe that we must prove something here assuming that the prover's claim is wrong, which in this case means that G_0 is not isomorphic to G_1. Now

assume that in one of the n iterations, P can answer both values of b with a permutations that satisfy V's check. Let ψ_0, ψ_1 be the permutations sent as response to $b = 0, 1$. Since V's checks are satisfied, we know that $\psi_0(H) = G_0$ and $\psi_1(H) = G_1$. It follows that G_0 is isomorphic to G_1 under the isomorphism $\psi_1\psi_0^{-1}$, a contradiction. Consequently, it must be the case that in all n iterations, the prover is able to answer at most one of the 2 possible values of b. Hence the probability of acceptance is at most 2^{-n}, which is certainly negligible in n.

Finally, let us show that the protocol is perfect zero-knowledge. To this end, we must build a simulator. The easiest way to think of a simulator usually is to think of it as an algorithm that tries to complete the protocol, playing the role of the prover, but of course without any special knowledge or computing power. Thus, a non-trivial trick is needed. In our case, we cannot just execute the protocol: we saw in the argument for soundness that knowing how to answer both of V's challenges at the same time implies we can compute an isomorphism between G_0 and G_1, and no efficient algorithm is known for this. However it *is* possible to prepare in such a way that one of the challenges can be answered. This is used in the following algorithm for a simulator M:

1. Start the machine V^*, which means giving it inputs G_0, G_1 (plus possibly some auxiliary input δ) and supplying random input bits for V^*. These are needed since V^* is allowed to be a probabilistic algorithm; we choose the random bits here and keep them fixed for the rest of the simulation.

2. To simulate one iteration, execute the following loop:

   (a) Choose a bit c and a permutation ψ at random. Set $H = \psi^{-1}(G_c)$ and send H to V^*.

   (b) Get b from V^*. If $b = c$, output H, b, ψ and exit the loop. Else, reset V^* to its state just before the last H was chosen, and go to step 2a.

   If we have completed simulation of all n iterations at this point, then stop. Else start at Step 2a again.

So in simulating one iteration, the simulator prepares to answer question c, and hopes that this is the question V^* will ask. If this happens, we're in business and can complete the simulation of the current iteration. Otherwise we just pretend the bad case never happened by rewinding V^* and then

we try again. At first sight, this rewinding technique can seem somewhat strange. However, it is essentially the same as rebooting your PC when it crashes: if we reach a configuration we don't like, we take the machine back to one we like better; so in this sense rewinding is an everyday experience[6].

To show that this simulator works, we need to show two things: M runs in expected polynomial time, and the distribution output by M is exactly the same as in a real protocol run.

Observe first, that by definition of zero-knowledge, we always prove correctness of a simulation assuming that P's claim is true, in our case this means that G_0 is isomorphic to G_1. Let S be the set of all graphs isomorphic to G_0 (or G_1). It is straightforward to check that the distribution of H generated in the simulation is the same as in the real protocol, namely the uniform distribution over S. Note that we need the assumption that G_0 is isomorphic to G_1 to conclude this.

In particular, the distribution of H is independent of c. It follows that the b chosen by V^* must be independent of c as well, and so $Prob(c = b) = 1/2$. Hence the expected number of times we do the loop to simulate one iteration is 2, and so the whole simulation takes expected time $2n$ times the time to go though the loop once, which is certainly polynomial in n.

Finally, the output distribution: The simulator produces in every iteration a triple (H, c, ψ), and then V^* outputs b, having seen H. First note that H is uniform over S, as it would be in a real conversation. It follows that the b sent by V^* is also distributed as in the real conversation (since V^* chooses b based only on its input and H). So the simulator can be described as follows: it keeps producing pairs of form (H, b), all of which are distributed as in the real protocol, until it happens to be the case that b equals the corresponding c-value, and then we select this as the pair (H, b) that will be used in the output. Now, by independency of H and c, the decision to keep H or rewind and throw it out does not depend on the choice of H. Hence the pair (H, b) we actually use will also be distributed as in the real protocol. And finally ψ is a random permutation mapping H to G_b, just as in the real protocol. Thus the output distribution of M matches the real protocol exactly.

This example demonstrates another basic design idea for zero-knowledge protocols: the prover is asked to answer one out of some set of questions. We set it up such that he can only answer all of them if his claim is true, but such that one can always prepare for answering any single question properly.

[6]If your PC never crashes, you should be making a fortune in consultancy instead of reading this note!

For other examples of this type of protocol, see e.g. [11, 12, 13, 21, 27, 33].

The principle behind the simulator for the graph isomorphism protocol is so general that it is worth while to describe it in general terms, making it directly usable in other protocols.

Suppose we are given a proof system (P, V) for language L. Suppose there exists a probabilistic poly-time machine M_L with the property that $(P, V) \sim^p M$ on input $x \in L$. Then M is called a perfect *honest-verifier simulator* (note that we use V and not an arbitrary V^* here).

Observe that what did in the simulation above was actually to exploit that the graph isomorphism protocol has such an honest-verifier simulator. Namely, we generate (H, c, ψ) by choosing c, ψ first and then computing H, and the "conversation" (H, c, ψ) is indeed distributed exactly as conversations between honest verifier end prover. Then what we did in the actual simulation could be described as: keep generating "honest-verifier" conversations (H, c, ψ), send H to V^*, get b back, and continue until it happens that $b = c$. Then output (H, b, ψ). This idea works in general:

Lemma 3.1 *The rewinding lemma: Let (P, V) be a proof system for language L, and let M be a perfect honest-verifier simulator for (P, V). Assume that conversations have the form (a, b, z), where P sends a, V responds with a random bit b, and P replies with z. Then (P, V) is perfect zero-knowledge.*

Exercise 8 Prove the rewinding lemma. Does the result also hold for statistical and computational zero-knowledge?

Exercise 9 [zero-knowledge proof of equivalence] Assume that we have a set $S \subseteq \{0,1\}^*$ and an equivalence relation $\sim$ on S such that $G, H \in S$ and $G \sim H$ implies that $|G| = |H|$. Your job is to construct a perfect zero-knowledge proof (with a polynomial time prover) where the common input is $G_0, G_1 \in S$ and the prover proves that $G_0 \sim G_1$. You are allowed to assume that there exist algorithms A, B and C with the following properties: 1) A takes three inputs (G, H, ϕ) and terminates with $b \in \{0,1\}$ in polynomial time in $|G|$; 2) If $G \not\sim H$, then $A(G, H, \phi) = 0$ for all $\phi \in \{0,1\}^*$; 3) If $G \sim H$ then there exists a unique $\phi \in \{0,1\}^*$ such that $A(G, H, \phi) = 1$ and $A(H, G, \phi) = 1$ (we call this ϕ *the witness* for $G \sim H$); 4) B takes one input G and terminates with $(H, \phi) \in S \times \{0,1\}^*$ in polynomial time in $|G|$; 5) If $(H, \phi) = B(G)$, then H is uniformly random in the equivalence class of G and ϕ is the witness for $G \sim H$; 6) C takes five inputs (G, H, I, ϕ, ψ) and terminates with $\pi \in \{0,1\}^*$ in polynomial time in $|G|$; 7) If ϕ is the witness for $G \sim H$ and ψ is the witness for $H \sim I$, then $\pi = C(G, H, I, \phi, \psi))$ is the witness for $G \sim I$.

Exercise 10 [zero-knowledge proof of identical discrete logarithm] Assume that we have a cyclic group G with prime order q. We can define an equivalence relation $\sim$ on $(G \setminus \{1\}) \times G$ where $(g_0, h_0) \sim (g_1, h_1)$ iff the discrete logarithm of h_0 base g_0 is equal to the discrete logarithm of h_1 base g_1, i.e. $(g_0, h_0) \sim (g_1, h_1)$ iff there exists $x \in \{0, 1, \ldots, q-1\}$ s.t. $h_0 = g_0^x$ and $h_1 = g_1^x$.

1. Observe that the equivalence class of (g_0, h_0) is exactly the $q-1$ pairs $(g_1, h_1) = (g_0^\pi, h_0^\pi)$ for $\pi \in \{1, \ldots, q-1\}$. Call $\pi \in \{1, \ldots, q-1\}$ for which $(g_1, h_1) = (g_0^\pi, h_0^\pi)$ a witness for $(g_0, h_0) \sim (g_1, h_1)$.

2. Use 1 to construct a perfect zero-knowledge proof (with a polynomial time prover) where the common input is $(g_0, h_0), (g_1, h_1) \in (G \setminus \{1\}) \times G$ and the prover proves that $(g_0, h_0) \sim (g_1, h_1)$ given a witness π for $(g_0, h_0) \sim (g_1, h_1)$. [Hint: You may appeal to Exercise 9.]

3.5 Known General Results and Open Problems

Having seen a few examples of zero-knowledge proofs, it is natural to ask some more general questions:

- Which languages have interactive proofs?
- Which languages have (perfect/statistical) zero-knowledge interactive proofs?
- Can we compose several zero-knowledge protocols and obtain again a zero-knowledge protocol?

It turns out that the answers depend strongly on whether the prover (and cheating provers) are allowed infinite computing power, or only polynomial time, that is, if we are talking about proof systems or arguments.

3.5.1 Results on Interactive Proofs and Arguments

For an unbounded prover, the first question has been answered by Shamir [34], where we define $IP = \{L |\ L$ has an interactive proof system$\}$:

Theorem 3.2 *$IP = PSPACE$, i.e. the statements that an all powerful prover can prove to a polynomially bounded verifier, are precisely those that can be verified using polynomially bounded memory (but possibly unbounded time).*

If the prover is polynomially bounded, it is clear that his only possible advantage over the verifier is that he may have more information than the verifier. In this case, the best the prover can do to convince the verifier is to simply send his information, s, say, to the verifier who should then be able to check the prover's statement based on s, where some error probability is allowed. The class of languages allowing such probabilistic verification of membership given auxiliary knowledge is already well known as $NBPP$ or MA. So if we define Bounded-ProverIP to be the class of languages that have interactive arguments, then we have:

Theorem 3.3 *Bounded-Prover*$IP = MA$

3.5.2 Results on Zero-Knowledge

We first look at the case of zero-knowledge interactive proofs. Let

$$ZKIP = \{L|\ L \text{ has a zero-knowledge interactive proof system}\}.$$

Goldreich, Micali and Wigderson [25] show that any $NP \subset ZKIP$ if commitment schemes with unconditional binding exist. This was extended to all of IP in [6]. This, together with Theorem 2.1 gives:

Theorem 3.4 *If one-way functions exist, then* $ZKIP = IP$.

It is natural to ask also about statistical and perfect zero-knowledge. Let $PZKIP$, $SZKIP$ denote the classes of languages with perfect, resp. statistical zero-knowledge proof systems. Except for the trivial $PZKIP \subset SZKIP \subset ZKIP$, very little is known with certainty. We know that a few languages with nice algebraic properties, such as graph isomorphism and quadratic residuosity[7] are in $PZKIP$. Also the complements of these languages are in $PZKIP$, and this is interesting since a problem such as graph non-isomorphism is not known to be in NP, and so it seems unlikely that $PZKIP \subset NP$ or $SKZIP \subset NP$. It also seems unlikely that the converse inclusion holds: Fortnow [20] has proved that if it does, then the polynomial hierarchy collapses - something believed to be false by many complexity theorists. In fact this can be seen as evidence that the graph isomorphism problem is *not* NP-complete, one of the few real evidences that have been found.

A nice characterisation of languages in $PZKIP$ or $SZKIP$ is an interesting open problem. We do know, however, some information on complete

[7] This is the set of pairs of numbers n, a, where a is a square modulo n

problems in $SZKIP$ [36], and that a proof system that is statistical zero-knowledge w.r.t. the honest verifier implies existence of a proof system that is statistical zero-knowledge in general [23].

Let us mention also a variant of the zero-knowledge concept, known as *non-interactive zero-knowledge*. In the non-interactive zero-knowledge model, an unbounded prover and a polynomial time verifier share access to a random string α. It is assumed as a part of the model, that α contains independent random bits. The prover must now convince the verifier that a common input x is in some language L by sending only 1 message σ (hence the "non-interactiveness"). The verifier then checks σ against x and α and accepts or rejects.

This proof system is called sound if whenever $x \notin L$, no prover can make the verifier accept with non-negligible probability over the choice of α. It is zero-knowledge if the pair σ, α can be simulated with an indistinguishable distribution in expected polynomial time.

This model was introduced by Blum, de Santis, Micali and Persiano [7] to formalise the absolute minimal amount of interaction required to prove non-trivial statements in zero-knowledge.

To distinguish between all the relevant complexity classes now involved, we use the following notation: Let $NIZK$, $NIPZK$ and $NISZK$ denote the classes of languages with non-interactive computational, perfect and statistical zero-knowledge proof systems.

Lapidot and Shamir [28] have shown that

Theorem 3.5 *If one-to-one surjective one-way functions exist, then* $NP \subset NIZK$.

It is an open question whether any one-way function would be sufficient.

The non-interactive model is weaker than the normal interactive model in that interaction is not allowed, but in another respect stronger because a random string with correct distribution is assumed to be given "for free". It is therefore not immediately clear whether any language that has a non-interactive zero-knowledge proof system also has an interactive one and vice versa. In [16], Damgård shows:

Theorem 3.6 *We have that* $NIZK \subset ZKIP$, $NISZK \subset SKZIP$ *and that* $NIPZK \subset PZKIP$.

We already know that if one-way functions exist, $ZKIP = PSPACE$. This together with the fact that a non-interactive proof uses only a constant

number of rounds provides very strong evidence that the first containment above is proper, since it is extremely unlikely that a constant number of rounds would be sufficient to prove all of IP. On the other hand, the corresponding questions for the classes where statistical or perfect zero-knowledge are required seem more open.

For the interactive argument model - which is the most interesting one in practice - the situation is again quite different. We have already seen that the only statements we can hope to prove at all are those in the class MA.

So the remaining question is whether we can prove any such statement in zero-knowledge, or even in perfect zero-knowledge.

In [4], Brassard Chaum and Crépeau show that any MA-language has a statistical (resp. perfect) zero-knowledge argument, if commitment schemes with unconditional (resp. perfect) hiding exist. This and the theory results for commitment schemes imply that

Theorem 3.7 *If one-way functions exist, then any language in MA has a statistical zero-knowledge interactive argument.*

Note that there is no conflict between this result and that of Fortnow mentioned above: Fortnow's result talks only about interactive proofs (and not arguments).

The concrete protocol constructions used to prove that all NP problems have zero-knowledge proof systems and arguments are in fact also proofs of knowledge. So equally general results on proofs of knowledge follow immediately.

3.5.3 On Composition of Zero-Knowledge Protocols

In general, the *sequential* composition of two zero-knowledge protocols is again zero-knowledge. An example of this is the graph isomorphism protocol shown above – it is in fact the result of repeating sequentially a basic step several times, where each step is zero-knowledge.

However, if we try doing the repetitions in parallel, then the resulting protocol does not seem to be zero-knowledge: we would get a scenario where P would send many graphs $H_1, ..., H_n$ at once, V would send challenges $b_1, ...b_n$ and P would reply by $\psi_1, ..., \psi_n$. The resetting technique for simulation does not work anymore: we would be forced to try to guess in advance all the bits $b_1, ...b_n$, and it would take us expected exponential time before the guess was correct. The idea that doing the protocol in parallel is not zero-knowledge may seem counterintuitive at first sight: why should

doing it in parallel tell V more about an isomorphism between G_0 and G_1? The answer is that while it might in fact be true that V learns nothing that could help him to compute such an isomorphism, this is not enough for zero-knowledge which requires that V learns *nothing whatsoever* that he could not compute himself. Indeed if the verifier computes its challenge bits as a one-way function of the $H_1, ..., H_n$ received, then it seems that conversation itself would be a piece of information that is difficult for V to generate on his own.

This discussion does not prove that the parallel version of the graph isomorphism protocol is not zero-knowledge, only that the resetting technique will not work for simulating it. However, Goldreich and Krawczyk [24] have shown that there exist protocols that are zero-knowledge, but where the parallel composition provably is not zero-knowledge.

A more complicated scenario which has been considered very recently is that of *concurrent zero-knowledge* where we allow arbitrary interleaving of different instances of protocols, i.e. while P is running a protocol with V_1, it starts doing (the same or) a different protocol with V_2, etc. There is no a priori time ordering fixed between messages sent in different protocols. We can ask whether this entire interaction is simulatable. There are results about this indicating that many well known protocols fail to be zero-knowledge in such a scenario, however, there are also ways around this problem. More information on this can be found in the paper [19] by Dwork and Sahai, which also contains pointers to more material.

3.6 Applications of Zero-Knowledge

One basic application of zero-knowledge protocols that is important in theory as well as in practice is the usage of zero-knowledge protocols as subprotocols in larger constructions, this could be voting schemes, key distribution protocols, or in general any multiparty computation. If we do not want to assume existence of secure channels, such constructions are usually not possible in the first place unless one-way functions exist. This means that in building such protocols we can assume without loss of generality that $NP \subset ZKIP$. And so whenever a player A sends a message in a protocol he can convince anybody else in zero-knowledge that he has computed his message according to the rules in the protocol. This follows since if the computation A was supposed to do is feasible in the first place, then the claim that the message is correctly computed can be verified in polynomial time given all A's data, and so is an NP-statement.

It follows that we can automatically transform any protocol that is secure

assuming players follow the rules into one that is secure even if players deviate arbitrarily from the protocol. This observation was first made in [25].

This can be interesting in practice if the involved zero-knowledge proofs are efficient. However, this is not always the case if we are using the general theoretical results we have covered. While they show what is in principle possible, most of the actual protocol constructions occurring in the proofs of those results are not very attractive in practice.

As an example, we know that a zero-knowledge proof or argument can be given for any NP language, and this is proved by providing a zero-knowledge proof for an NP complete problem such as Boolean Circuit satisfiability (SAT). When we are given a concrete problem instance $x \in L$, where $L \in NP$, then to use the general result, we must first construct from x a Boolean circuit which is satisfiable precisely if $x \in L$, and then use the protocol for SAT.

This approach often results in very large circuits, for problem instances of interest in real life, typically at least 10.000 to 100.000 binary gates. It is therefore of interest to be able to construct instead an ad hoc zero-knowledge protocol for the problem in question, such as the graph isomorphism protocol above. A few problems are "nice" in this sense, in that they allow construction of particularly efficient protocols. This is often true of problems derived from number theory, and we mention some examples below. Still, there are also cases where the only solution we know is to use general techniques. This can be the case e.g. if P wants to show that for a given bit string y he knows x such that $h(x) = y$, where h is some cryptographic hash function. Since such functions are usually constructed deliberately to have none of the nice algebraic properties that enable efficient zero-knowledge directly, we have to resort to the general techniques. SAT is often the natural NP complete problem to use, so efficient zero-knowledge protocols for SAT are of particular interest. Results by Cramer and Damgård in this direction show that one can prove satisfiability of a Boolean circuit while communicating only a number of bit commitments linear in the size of the circuit [11]. Using preprocessing, one can even reduce the proof to one message containing 2 bits pr. gate in the circuit [12]. Thus, general techniques can in fact be practical in some cases.

Still, the largest potential for practical applications of zero-knowledge comes from extremely efficient protocols specially designed for particular problems such as the quadratic residuosity problem [21], the discrete logarithm problem [33], or the RSA root extraction problem[27]. The typical use here is for the classical user identification problem that we mentioned

earlier: each user U gets a solution to a hard problem instance x_U, and can identify himself by proving in zero-knowledge that he knows a solution to x_U. By the zero-knowledge property, none of the proofs conducted by U will help an adversary to find a solution to x_U. Still, by the soundness property, an adversary can only impersonate U if he can find a solution to x_U. So if he succeeds it means he could find a solution to x_U from scratch, and this is not possible if the underlying problem is hard. Using a secure hash function, one can also use these (interactive) identification protocols to build (non-interactive) signature schemes [21]. These can be more efficient than RSA signatures, but have so far only conjectured security in the sense that we do not know how to reduce the security to any well established computational assumption.

The most efficient versions of these protocols yield error probability exponentially small in the security parameter, even though the communication required is only linear. Unfortunately, these protocols are only zero-knowledge against the *honest* verifier, and hence have no provable security in real life. Feige and Shamir [22] point out a possible way around this problem: the identification scenario does not really require the full power of zero-knowledge. It is enough if the protocol does not help the verifier (or anyone else) to find the provers secret (while zero-knowledge ensures that the verifier learns nothing new whatsoever). This is so since we can show that an adversary needs to know the prover's secret to impersonate the prover. Protocols with this weaker property are called *Witness Hiding* (WH), and might conceivably be easier to construct. In [13] Cramer, Damgård and Schoenmakers show that the efficient honest verifier zero-knowledge protocols of [33, 27] can be transformed into WH protocols while preserving the efficiency.

The results just mentioned and many others in the area of efficient zero-knowledge and WH protocols revolve around protocols of a particular form where P sends a message, V sends a random challenge, and P gives an answer that can be checked by V (this is the form of the basic step in the graph isomorphism protocol). While such protocols by themselves have only limited security properties (e.g. they either have large error probability or are only honest verifier zero-knowledge), it turns out that they can be used in a modular way in a number of constructions of protocols and signature schemes with simultaneously high efficiency and provable security. For instance, a prover can show that he knows at least t out of $n > t$ secrets without revealing which t secrets is involved [13, 35]. This can be important, e.g. in protocols where anonymity is desired. For a nice introduction to this entire area, see [8].

References

[1] W.Alexi, B.Chor, O.Goldreich and C.P.Schnorr: *RSA and Rabin Functions: Certain parts are as hard as the Whole*, SIAM J.Computing, 17(1988), 194-209.

[2] M. Bellare and and O. Goldreich: *On Defining Proofs of Knowledge*, Proceedings of Crypto '92, Springer Verlag LNCS, vol. 740, pp. 390–420.

[3] M. Ben-Or, S. Goldwasser, A. Wigderson: *Completeness theorems for Non-Cryptographic Fault-Tolerant Distributed Computation*, Proc. ACM STOC '88, pp. 1–10.

[4] G. Brassard, D. Chaum and C. Crépeau: *Minimum Disclosure Proofs of Knowledge*, JCSS, vol.37, pp. 156–189, 1988.

[5] J.Brandt, I.Damgaård, P.Landrock and T.Pedersen: *Zero-Knowledge Authentication Scheme with Secret Key Exchange*, J.Cryptology, vol 11(1998), 147-160.

[6] M.Ben-Or, O.Goldreich, S.Goldwasser, J.Håstad, J.Kilian, S.Micali and P.Rogaway: *Everything Provable is Provable in Zero-Knowledge*, Proceedings of Crypto 88, Springer Verlag LNCS series, 37–56.

[7] Blum, De Santis, Micali and Persiano: *Non-Interactive Zero-Knowledge*, SIAM J.Computing, Vol.20, no.6, 1991.

[8] R.Cramer: *Modular Design of Secure, yet Practical Cryptographic Protocols*, PhD Thesis, University of Amsterdam 1996.

[9] C.Crépeau: *Efficient Cryptographic Protocols based on Noisy Channels*, Proceedings of EuroCrypt 97, Springer Verlag LNCS series, vol.1233, p.306-317.

[10] D. Chaum, C. Crépeau, I. Damgård: *Multi-Party Unconditionally Secure Protocols*, Proc. of ACM STOC '88, pp. 11–19.

[11] R. Cramer and I. Damgård: *Linear Zero-Knowledge*, Proc. of STOC 97.

[12] R. Cramer and I. Damgård: *Zero-Knowledge Proofs for Finite Field Arithmetic; or Can Zero-Knowldge be for Free?*, Proceedings of Crypto 98, Springer Verlag LNCS series.

[13] R. Cramer, I. Damgård and B. Schoenmakers: *Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols*, Proceedings of Crypto '94, Springer verlag LNCS, vol. 839, pp. 174–187.

[14] C.Crépeau and J.Kilian:*Achieving Oblivious Transfer using Weakened Security Assumptions*, Proc. of FOCS 88, p.42-52.

[15] I.Damgård: *Collision Free Hash Functions and Public Key Signature Schemes*, Proc. of EuroCrypt 87, Springer Verlag LNCS series.

[16] I.Damgård: *Interactive Hashing can Simplify Zero-Knowledge Protocol Design Without Computational Assumptions*, Proc. of Crypto 93, Springer Verlag LNCS series.

[17] I. Damgård and B. Pfitzmann: *Sequential Iteration of Interactive Arguments*, Proc. of ICALP 98, Springer Verlag LNCS series.

[18] I. Damgaård, B. Pfitzmann and T.Pedersen: *Statsitical Secrecy and Multi-Bit Commitments*, IEEE Trans.Info.Theory, vol.44 (1998), 1143-1151.

[19] C.Dwork and A.Sahai: *Concurrent Zero-Knowledge: Reducing the Need for Timing Constraints*, Proc. of Crypto '98, Springer Verlag LNCS series.

[20] L.Fortnow: *The complexity of Perfect Zero-Knowledge*, Adv. in Computing Research, vol.5, 1989, 327–344.

[21] A.Fiat, A.Shamir: *How to Prove Yourself, Practical Solutions to Identification and Signature Problems*, proc. of Crypto 86, Springer Verlag LNCS series.

[22] U. Feige and A. Shamir: *Witness Indistinguishable and Witness Hiding Protocols*, Proc. of STOC '90.

[23] O.Goldreich, A.Sahai, S.Vadhan: *Honest-Verifier Statistical Zero-Knowledge Equals General Statistical Zero-Knowledge*, Proc. of STOC '98.

[24] O. Goldreich and A. Kahan: *How to Construct Constant-Round Zero-Knowledge Proof Systems for NP*, Journal of Cryptology, (1996) 9: 167–189.

[25] O. Goldreich, S. Micali and A. Wigderson: *Proofs that yield Nothing but their Validity and a Methodology of Cryptographic Protocol Design*, Proceedings of FOCS '86, pp. 174–187.

[26] S. Goldwasser, S. Micali and C. Rackoff: *The Knowledge Complexity of Interactive Proof Systems*, SIAM J.Computing, Vol. 18, pp. 186-208, 1989.

[27] L.Guillou and J.J.Quisquater: *A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing both Transmission and Memory*, Proc. of EuroCrypt 88, Springer Verlag LNCS 330.

[28] Lapidot and Shamir: *Publicly Verifiable Non-Interactive Zero-Knowledge Proofs*, Proc. of Crypto 90, Springer Verlag LNCS series.

[29] M.Naor: *Bit Commitment using pseudo-randomness*, Proceedings of Crypto 89, Springer Verlag LNCS series.

[30] M.Naor, R.Ostrovsky, S.Venkatesan, M.Yung: *Perfect Zero-Knowledge Arguments for NP using Any One-Way Permutation*, J.Cryptology, vol.11 (1998), 87-108.

[31] M.Naor, M.Yung: *Universal One-Way hash Functions and their Cryptographic Applications*, Proc. of STOC '89, 33-43.

[32] Iftach Haitner and Omer Reingold: *Statistically-Hiding Commitment from Any One-Way Function*, the Eprint archive, www.iacr.org.

[33] C. P. Schnorr: *Efficient Signature Generation by Smart Cards*, Journal of Cryptology, 4 (3): 161–174, 1991.

[34] A.Shamir: *IP=PSPACE*, Journal of the ACM, vol.39 (1992), 869-877.

[35] A.De Santis, G.Crescenzo, G.Persiano, M.Yung: *On Monotone Formula Closure of SZK*, proc. of FOCS '94.

[36] A.Sahai and S.Vadhan: *A Complete Promise Problem for Statistical Zero-Knowledge*, Proc. of FOCS '97.