CHAPTER 21

Zero-Knowledge Proofs

Chapter Goals

- To introduce zero-knowledge proofs.
- To explain the notion of simulation.
- To introduce Sigma protocols.
- To explain how these can be used in a voting protocol.

21.1. Showing a Graph Isomorphism in Zero-Knowledge

Suppose Alice has a password and wants to log in to a website run by Bob, but she does not quite trust the computer Bob is using to verify the password. If she just sends the password to Bob then Bob's computer will learn the whole password. To get around this problem one often sees websites that ask for the first, fourth and tenth letter of a password one time, and then maybe the first, second and fifth the second time and so on. In this way Bob's computer only learns three letters at a time. So the password can be checked but in each iteration of checking only three letters are leaked. It clearly would be better if Bob could verify that Alice has the password in such a way that Alice never has to reveal *any* of the password to Bob. This is the problem this chapter will try to solve.

So we suppose that Alice wants to convince Bob that she knows something without Bob finding out exactly what Alice knows. This apparently contradictory state of affairs is dealt with using zero-knowledge proofs. In the literature of zero-knowledge proofs, the role of Alice is called the prover, since she wishes to prove something, whilst the role of Bob is called the verifier, since he wishes to verify that the prover actually knows something. Often, and we shall also follow this convention, the prover is called Peggy and the verifier is called Victor.

The classic example of a zero-knowledge proof is based on the graph isomorphism problem. Given two graphs G_1 and G_2, with the same number of vertices, we say that the two graphs are isomorphic if there is a relabelling (i.e. a permutation) of the vertices of one graph which produces the second graph. This relabelling ϕ is called a graph isomorphism, which is denoted by

$$\phi : G_1 \longrightarrow G_2.$$

It is a computationally hard problem to determine a graph isomorphism between two graphs. As a running example consider the two graphs in Figure 21.1, linked by the permutation $\phi = (1, 2, 4, 3)$.

Suppose Peggy knows the graph isomorphism ϕ between two public graphs G_1 and G_2, so we have $G_2 = \phi(G_1)$. We call ϕ the prover's private input, whilst the graphs G_1 and G_2 are the public or common input. Peggy wishes to convince Victor that she knows the graph isomorphism, without revealing to Victor the precise nature of the graph isomorphism. This is done using the following zero-knowledge proof.

N.P. Smart, *Cryptography Made Simple*, Information Security and Cryptography, DOI 10.1007/978-3-319-21936-3_21

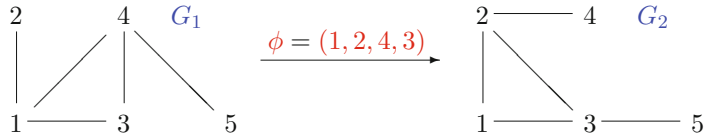

FIGURE 21.1. Example graph isomorphism

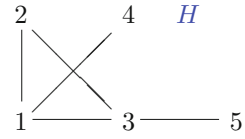

FIGURE 21.2. Peggy's committed graph

- Peggy takes the graph G_2 and applies a secret random permutation ψ to the vertices of G_2 to produce another isomorphic graph $H \leftarrow \psi(G_2)$. In our running example we take $\psi = (1, 2)$; the isomorphic graph H is then given by Figure 21.2.
- Peggy now publishes H as a **commitment**. She of course knows the following secret graph isomorphisms

$$\begin{aligned} \phi &: G_1 \longrightarrow G_2, \\ \psi &: G_2 \longrightarrow H, \\ \psi \circ \phi &: G_1 \longrightarrow H. \end{aligned}$$

- Victor now gives Peggy a **challenge**. He selects[1] $b \in \{1, 2\}$ and asks for the graph isomorphism between G_b and H
- Peggy now gives her **response** by returning either $\chi = \psi$ or $\chi = \psi \circ \phi$, depending on the value of b.
- Victor now verifies whether $\chi(G_b) = H$.

The transcript of the protocol then looks like

$$\begin{aligned} P &\longrightarrow V : H, \\ V &\longrightarrow P : b, \\ P &\longrightarrow V : \chi. \end{aligned}$$

In our example if Victor chooses $b = 2$ then Peggy simply needs to publish ψ. However, if Victor chooses $b = 1$ then Peggy publishes

$$\psi \circ \phi = (1, 2) \circ (1, 2, 4, 3) = (2, 4, 3).$$

We can then see that $(2, 4, 3)$ is the permutation which maps graph G_1 onto graph H. But to compute this we needed to know the hidden isomorphism ϕ. Thus when $b = 2$ Victor is checking whether Peggy is honest in her commitment, whilst if $b = 1$ he is checking whether Peggy is honest in her claim to know the isomorphism from G_1 to G_2.

If Peggy does not know the graph isomorphism ϕ then, before Victor gives his challenge, she will need to know the graph G_b which Victor is going to pick. Hence, if Peggy is cheating she will

[1] Since it is seleted by Victor we denote the value b in blue.

only be able to respond to correctly to Victor fifty percent of the time. So, repeating the above protocol a number of times, a non-cheating Peggy will be able to convince Victor that she really does know the graph isomorphism, with a small probability that a cheating Peggy will be able to convince Victor incorrectly.

Now we need to determine whether Victor learns anything from running the protocol, i.e. is Peggy's proof really zero-knowledge? We first notice that Peggy needs to produce a different value of H on every run of the protocol, otherwise Victor can trivially cheat. We assume therefore that this does not happen.

One way to see whether Victor has learnt something after running the protocol is to look at the transcript of the protocol and ask after having seen the transcript whether Victor has gained any knowledge, or for that matter whether anyone looking at the protocol but not interacting learns anything. One way to see that Victor has not learnt anything is to see that Victor could have written down a valid protocol transcript without interacting with Peggy at all. Hence, Victor cannot use the protocol transcript to convince someone else that he knows Peggy's secret isomorphism. He cannot even use the protocol transcript to convince another party that Peggy knows the secret graph isomorphism ϕ.

Victor can produce a valid protocol transcript using the following *simulation*:

- $b \leftarrow \{1, 2\}$.
- Generate a random isomorphism χ of the graph G_b to produce the graph H.
- Output the transcript

$$\begin{aligned} P &\longrightarrow V : H, \\ V &\longrightarrow P : b, \\ P &\longrightarrow V : \chi. \end{aligned}$$

Hence, the interactive nature of the protocol is what provides the "proof" in the zero-knowledge proof. We remark that the three-pass system of

$$\text{commitment} \longrightarrow \text{challenge} \longrightarrow \text{response}$$

is the usual characteristic of such protocols when deployed in practice. Notice how this is similar to the signature schemes we discussed in Section 16.5.4. This is not coincidental, as we shall point out below.

Clearly two basic properties of an interactive proof system are

- **Completeness:** If Peggy really knows the thing being proved and follows the protocol, then Victor should accept her proof with probability one.
- **Soundness:** If Peggy does not know the thing being proved, then whatever she does, Victor should only have a small probability of actually accepting the proof.

Just as with commitment schemes we can divide zero-knowledge protocols into categories depending on whether they are secure with respect to computationally bounded or unbounded adversaries. We usually assume that Victor is a polynomially bounded party, whilst Peggy is unbounded[2]. In the above protocol based on graph isomorphism we saw that the soundness probability was equal to one half. Hence, we needed to repeat the protocol a number of times to improve this to something close to one.

The zero-knowledge property we have already noted is related to the concept of a simulation. Suppose the set of valid transcripts (produced by true protocol runs) is denoted by $\mathcal{V}$, and let the set of possible simulations be denoted by $\mathcal{S}$. The security is therefore related to how much the set $\mathcal{V}$ is like the set $\mathcal{S}$. A zero-knowledge proof is said to have perfect zero-knowledge if the two

[2]Although in many of our examples the existence of a witness is certain and hence we might as well assume that Peggy knows the witness already and is bounded.

sets $\mathcal{V}$ and $\mathcal{S}$ are essentially identical, in which case we write $\mathcal{V} = \mathcal{S}$. If the two sets have "small" statistical distance[3], but cannot otherwise be distinguished by an all-powerful adversary, we say we have statistical zero-knowledge, and we write $\mathcal{V} \approx_s \mathcal{S}$. If the two sets are only indistinguishable by a computationally bounded adversary we say that the zero-knowledge proof has computational zero-knowledge, and we write $\mathcal{V} \approx_c \mathcal{S}$.

21.2. Zero-Knowledge and $\mathcal{NP}$

So the question arises as to what can be shown in zero-knowledge. Above we showed that the knowledge of whether two graphs are isomorphic can be shown in zero-knowledge. Thus the decision problem of `Graph Isomorphism` lies in the set of all decision problems which can be proven in zero-knowledge. But `Graph Isomorphism` is believed to lie between the complexity classes $\mathcal{P}$ and $\mathcal{NP}$-complete, i.e. it can neither be solved in polynomial time, nor is it $\mathcal{NP}$-complete.

We can think of $\mathcal{NP}$ problems as those problems for which there is a witness (or proof) which can be produced by an all-powerful prover, but for which a polynomially bounded verifier is able to verify the proof. However, for the class of $\mathcal{NP}$ problems the prover and the verifier do not interact, i.e. the proof is produced and then the verifier verifies it.

If we allow interation then something quite amazing happens. Consider an all powerful prover who interacts with a polynomially bounded verifier. We wish the prover to convince the verifier of the validity of some statement. This is exactly what we had in the previous section except that we only require the completeness and soundness properties, i.e. we do not require the zero-knowledge property. The decision problems which can be proven to be true in such a manner form the complexity class of *interactive proofs*, or $\mathcal{IP}$. It can be shown that the complexity class $\mathcal{IP}$ is equal to the complexity class $\mathcal{PSPACE}$, i.e. the set of all decision problems which can be solved using polynomial space. It is widely believed that $\mathcal{NP} \subsetneq \mathcal{PSPACE}$, which implies that having interaction really gives us something extra.

So what happens to interactive proofs when we add the zero-knowledge requirement? We can define a complexity class $\mathcal{CZK}$ of all decision problems whose solutions can be verified using a computational zero-knowledge proof. We have already shown that the problem of `Graph Isomorphism` lies in $\mathcal{CZK}$, but this might not include all of the $\mathcal{NP}$ problems. However, since 3-colourability is $\mathcal{NP}$-complete, we have the following elegant proof that $\mathcal{NP} \subset \mathcal{CZK}$,

Theorem 21.1. *The problem of 3-colourability of a graph lies in $\mathcal{CZK}$, assuming a computationally concealing commitment scheme exists.*

Proof. Consider a graph $G = (V, E)$ in which the prover knows a colouring ψ of G, i.e. a map $\psi : V \to \{1, 2, 3\}$ such that $\psi(v_1) \neq \psi(v_2)$ if $(v_1, v_2) \in E$. The prover first selects a commitment scheme $C(x, r)$ and a random permutation π of the set $\{1, 2, 3\}$. Note that the function $\pi(\psi(v))$ defines another three-colouring of the graph. Now the prover commits to this second three-colouring by sending to the verifier the commitments

$$c_i = C\left(\pi(\psi(v_i)), r_i\right) \text{ for all } v_i \in V.$$

The verifier then selects a random edge $(v_i, v_j) \in E$ and sends this to the prover. The prover now decommits to the commitment, by returning the values of

$$\pi(\psi(v_i)) \text{ and } \pi(\psi(v_j)),$$

and the verifier checks that

$$\pi(\psi(v_i)) \neq \pi(\psi(v_j)).$$

We now turn to the three required properties of a zero-knowledge proof.

[3]See Chapter 7.

Completeness: The above protocol is complete since any valid prover will get the verifier to accept with probability one.
Soundness: If we have a cheating prover, then at least one edge is invalid, and with probability at least $1/|E|$ the verifier will select an invalid edge. Thus with probability at most $1 - 1/|E|$ a cheating prover will get a verifier to accept. By repeating the above proof many times one can reduce this probability to as low a value as we require.
Zero-Knowledge: Assuming the commitment scheme is computationally concealing, the obvious simulation and the real protocol will be computationally indistinguishable. □

Notice that this is a very powerful result. It says that virtually any statement which is likely to come up in cryptography can be proved in zero-knowledge. Clearly the above proof would not provide a practical implementation, but at least we know that very powerful tools can be applied. In the next section we turn to proofs that can be applied in practice. But before doing that we note that the above result can be extended even further.

Theorem 21.2. *If one-way functions exist then* $\mathcal{CZK} = \mathcal{IP}$*, and hence* $\mathcal{CZK} = \mathcal{PSPACE}$*.*

21.3. Sigma Protocols

One can use a zero-knowledge proof of possession of some secret as an identification scheme. The secret in the identification scheme will be the hidden information, or witness, e.g. the graph isomorphism in our previous example. Then we use the zero-knowledge protocol to prove that the person knows the isomorphism, without revealing anything about it. The trouble with the above protocol for graph isomorphisms is that it is not very practical. The data structures required are very large, and the protocol needs to be repeated a large number of times before Victor is convinced that Peggy really knows the secret.

This is exactly what a Sigma protocol provides. It is a three-move protocol: the prover goes first (in the commitment phase), then the verifier responds (with the challenge), and finally the prover provides the final response; the verifier is then able to verify the proof. This is exactly like our graph isomorphism proof earlier. But for Sigma protocols we make some simplifying assumptions; in particular we assume that the verifier is honest (in that he will always follow the protocol correctly).

21.3.1. Schnorr's Identification Protocol: In essence we have already seen a Sigma protocol which has better bandwidth and error properties when we discussed Schnorr signatures in Chapter 16. Suppose Peggy's secret is now the discrete logarithm x of y with respect to g in some finite abelian group G of prime order q. To create an identification protocol, we want to show in zero-knowledge that Peggy knows the value of x. The protocol for proof of knowledge now goes as follows

$$\begin{aligned} &P \longrightarrow V : r \leftarrow g^k \text{ for a random } k \leftarrow \mathbb{Z}/q\mathbb{Z}, \\ &V \longrightarrow P : e \leftarrow \mathbb{Z}/q\mathbb{Z}, \\ &P \longrightarrow V : s \leftarrow k + x \cdot e \pmod{q}. \end{aligned}$$

Victor now verifies that Peggy knows the secret discrete logarithm x by verifying that $r = g^s \cdot y^{-e}$. Let us examine this protocol in more detail.

Completeness: We first note that the protocol is complete, in that if Peggy actually knows the discrete logarithm then Victor will accept the protocol since

$$g^s \cdot y^{-e} = g^{k+x\cdot e} \cdot (g^x)^{-e} = g^k = r.$$

Soundness: If Peggy does not know the discrete logarithm x then one can informally argue that she will only be able to cheat with probability $1/q$, which is much better than the $1/2$ from the earlier graph-isomorphism-based protocol. We can however show that the protocol has something called the special soundness property.

Definition 21.3 (Special Soundness)**.** *Suppose that we have two protocol runs with transcripts* (r, e, s) *and* (r, e', s')*. Note that the commitments are equal but that the challenges (and hence responses) are different. A protocol is said to have the special soundness property if given two such transcripts we can recover* x*.*

As an example, for our Schnorr protocol above, we have that given two such verifying transcripts we have that

$$r = g^s \cdot y^{-e} = g^{s'} \cdot y^{-e'} = r.$$

This implies in turn that

$$s + x \cdot (-e) = s' + x \cdot (-e') \pmod{q}.$$

Hence, we recover x via

$$x \leftarrow \frac{s - s'}{e - e'} \pmod{q}.$$

Notice that this proof of soundness is almost exactly the same as our use of the forking lemma to show that Schnorr signatures are EUF-CMA secure assuming discrete logarithms are hard. The above algorithm, which takes (r, e, s) and (r, e', s') and outputs the discrete logarithm x, is a knowledge extractor as defined below. It is the existence of this algorithm which shows we have a zero-knowledge proof of knowledge, as defined below, and not just a zero-knowledge proof.

More formally, suppose we have a statement, say $X \in \mathcal{L}$, where $\mathcal{L}$ is some language in $\mathcal{NP}$. Since the language lies in $\mathcal{NP}$ we know there exists a witness w. Now a zero-knowledge proof is an interactive protocol which given a statement $X \in \mathcal{L}$ will allow *an infinitely powerful prover* to demonstrate that $X \in \mathcal{L}$. Note here that the prover may not actually "know" the witness. However, a protocol is said to be a zero-knowledge proof of knowledge if it is a zero-knowledge proof and there exists an algorithm, called a knowledge extractor E, which can use a valid prover to output the witness w.

In our above example we take the prover and run her once, then rewind her to the point when she asks for the verifier's challenge, we then supply her with another challenge and thus end up obtaining the two tuples (r, e, s) and (r, e', s'). Then the special soundness implies that there is a knowledge extractor E which takes as input (r, e, s) and (r, e', s') and outputs the witness. We write $x \leftarrow E\left((r, e, s), (r, e', s')\right)$.

Zero-Knowledge: But does Victor learn anything from the protocol? The answer to this is no, since Victor could simulate the whole transcript in the following way.

- $e \leftarrow \mathbb{Z}/q\mathbb{Z}$.
- $r \leftarrow g^s \cdot y^{-e}$.
- Output the transcript

$$\begin{aligned} P &\longrightarrow V : r, \\ V &\longrightarrow P : e, \\ P &\longrightarrow V : s. \end{aligned}$$

In other words the protocol is zero-knowledge, in that someone cannot tell the simulation of a transcript from a real transcript. This is exactly the same simulation we used when simulating the signing queries in our proof of security of Schnorr signatures in Theorem 16.12.

21.3.2. Formalizing Sigma Protocols: Before we discuss other Sigma protocols, we introduce some notation to aid our discussion. We assume we have some statement $X \in \mathcal{L}$, for some $\mathcal{NP}$ language $\mathcal{L}$, and we want to prove that the prover "knows" the underlying witness w.

Suppose we wish to prove knowledge of the variable x via a Sigma protocol, then

- $R(w, k)$ denotes the algorithm used to compute the commitment r, where k is the random value used to produce the commitment.
- e is the challenge from a set $\mathbb{E}$.
- $S(e, w, k)$ denotes the algorithm which the prover uses to compute her response $s \in \mathbb{S}$ given e.
- $V(r, e, s)$ denotes the verification algorithm.
- $S'(e, s)$ denotes the simulator's algorithm which creates a value of a commitment r which will verify the transcript (r, e, s), for random input values $e \in \mathbb{E}$ and s.
- $E\left((r, e, s), (r, e', s')\right)$ is the knowledge extractor which will output w.

All algorithms are assumed to implicitly have as input the public statement X for which w is a witness. Using this notation Schnorr's identification protocol becomes the following. The statement X is that

$$\exists x \text{ such that } y = g^x,$$

and the witness is $w = x$. We then have

$$\begin{aligned} R(x, k) &:= r \leftarrow g^k, \\ S(e, x, k) &:= s \leftarrow k + e \cdot x \pmod{q}, \\ V(r, e, s) &:= \textbf{true} \text{ if and only if } (r = g^s \cdot y^{-e}), \\ S'(e, s) &:= r \leftarrow g^s \cdot y^{-e}, \\ E((r, e, s), (r, e', s')) &:= x \leftarrow \frac{s - s'}{e - e'} \pmod{q}. \end{aligned}$$

21.3.3. Associated Identification Protocol: We can create an identification protocol from *any* Sigma protocol as follows: We have some statement X which is bound to an entity P such that P has been given the witness w for the statement being valid. To prove that P really does know w without revealing anything about w, we execute the following protocol:

$$\begin{aligned} P &\longrightarrow V : r \leftarrow R(w, k), \\ V &\longrightarrow P : e \leftarrow \mathbb{E}, \\ P &\longrightarrow V : s \leftarrow S(e, w, k), \end{aligned}$$

where the verifier accepts the claimed identity if and only if $V(r, e, s)$ returns **true**. One can think of w in this protocol as a very strong form of "password" authentication, where no information about the "password" is leaked. Of course this will only be secure if finding the witness given only the statement is a hard problem, since otherwise anyone could compute the witness on their own.

21.3.4. Associated Signature Schemes: We have already seen how the Schnorr signature scheme and Schnorr's identification protocol are related. It turns out that *any* Sigma protocol with the special soundness property can be turned into a digital signature scheme. The method of transformation is called the Fiat–Shamir heuristic, since it only produces a heuristically secure scheme as the security proof requires the use of the random oracle H whose codomain is the set $\mathbb{E}$.

- **Key Generation:** The public key is the statement $X \in \mathcal{L}$, and the secret key is the witness w.

- **Signing:** The signature on a message m is generated by
$$r \leftarrow R(w,k),$$
$$e \leftarrow H(r\|m),$$
$$s \leftarrow S(e,w,k).$$
Output (e,s) as the signature.
- **Verification:** Generate $r' \leftarrow S'(e,s)$, and then check whether $e = H(r'\|m)$.

Using the same technique as in the proof of Theorem 16.12 we can prove the following.

Theorem 21.4. *In the random oracle model let A denote an* EUF-CMA *adversary against the above signature scheme with advantage ϵ, making q_H queries to its hash function H. Then there is an adversary B against the associated Sigma protocol which given $X \in \mathcal{L}$ can extract the witness w with advantage ϵ' such that*
$$\epsilon' \geq \frac{\epsilon^2}{q_H} - \frac{\epsilon}{q_H}.$$

PROOF. The proof is virtually identical to that of Theorem 16.12. We take the adversary A and wrap it inside another algorithm A' which does not make queries to a signature oracle. This is done using the simulator S' for the Sigma protocol. We then apply the forking lemma to A' in order to construct an algorithm B which will output a pair of tuples (r,e,s) and (r,e',s'). We then pass these tuples to the knowledge extractor E in order to recover the witness w. □

21.3.5. Non-interactive Proofs: Sometimes we want to prove something in zero-knowledge but not have the interactive nature of a protocol. For example one entity may be sending some encrypted data to another entity, but wants to prove to anyone seeing the ciphertext that it encrypts a value from a given subset. If a statement can be proved with a Sigma protocol we can turn it into a non-interactive proof by replacing the verifier's challenge component with a hash of the *commitment and the statement*. This last point is often forgotten, since it is not needed in the signature example above, and hence people forget about it when producing general non-interactive proofs. Hence, in the Schnorr proof of knowledge of discrete logarithms protocol we would define the challenge as
$$e \leftarrow H(r\|g\|y).$$

21.3.6. Chaum–Pedersen Protocol: We now present a Sigma protocol called the Chaum–Pedersen protocol which was first presented in the context of electronic cash systems, but which has very wide application. Suppose Peggy wishes to prove she knows two discrete logarithms
$$y_1 = g^{x_1} \text{ and } y_2 = h^{x_2}$$
such that $x_1 = x_2$, i.e. we wish to present not only a proof of knowledge of the discrete logarithms, but also a proof of equality of the hidden discrete logarithms. We assume that g and h generate groups of prime order q, and we denote the common discrete logarithm by x to ease notation. Using our prior notation for Sigma protocols, the Chaum–Pedersen protocol can be expressed via
$$\begin{aligned}
R(x,k) &:= (r_1,r_2) \leftarrow (g^k, h^k),\\
S(e,x,k) &:= s \leftarrow k - e\cdot x \pmod q,\\
V((r_1,r_2),e,s) &:= \textbf{true} \text{ if and only if } (r_1 = g^s \cdot y_1^e \text{ and } r_2 = h^s \cdot y_2^e),\\
S'(e,s) &:= (r_1,r_2) \leftarrow (g^s \cdot y_1^e, h^s \cdot y_2^e),\\
E((r,e,s),(r,e',s')) &:= x \leftarrow \frac{s'-s}{e-e'} \pmod q.
\end{aligned}$$
Note how this resembles two concurrent runs of the Schnorr protocol, but with a single challenge value. The Chaum–Pedersen protocol is clearly both complete and has the zero-knowledge property,

the second fact follows since the simulation $S'(e, s)$ produces transcripts which are indistinguishable from a real transcript.

We show it is sound, by showing it has the special soundness property. Hence, we assume two protocol runs with the same commitments (r_1, r_2), but with different challenges e and e', and corresponding valid responses s and s'. With this data we need to show that this reveals the common discrete logarithm via the extractor E, *and* that the discrete logarithm is indeed common. Since the two transcripts pass the verification test we have that

$$r_1 = g^s \cdot y_1^e = g^{s'} \cdot y_1^{e'} \text{ and } r_2 = h^s \cdot y_2^e = h^{s'} \cdot y_2^{e'}.$$

But this implies that ${y_1}^{e-e'} = g^{s'-s}$ and ${y_2}^{e-e'} = h^{s'-s}$, and so

$$(e - e') \cdot \text{dlog}_g(y_1) = s' - s \text{ and } (e - e') \cdot \text{dlog}_h(y_2) = s' - s.$$

Hence, the two discrete logarithms are equal and can be extracted from

$$x = \frac{s' - s}{e - e'} \pmod{q}.$$

21.3.7. Proving Knowledge of Pedersen Commitments: Often one commits to a value using a commitment scheme, but the receiver is not willing to proceed unless one proves one knows the value committed to. In other words the receiver will only proceed if he knows that the sender will at some point *be able* to reveal the value committed to. For the commitment scheme

$$B(x) = g^x$$

this is simple, we simply execute Schnorr's protocol for proof of knowledge of a discrete logarithm. For Pedersen commitments $B_a(x) = h^x \cdot g^a$ we need something different. In essence we wish to prove knowledge of x_1 and x_2 such that

$$y = {g_1}^{x_1} \cdot {g_2}^{x_2}$$

where g_1 and g_2 are elements in a group of prime order q. We note that the following protocol generalizes easily to the case when we have more bases, i.e.

$$y = {g_1}^{x_1} \cdots {g_n}^{x_n},$$

a generalization that we leave as an exercise. In terms of our standard notation for Sigma protocols we have

$$\begin{aligned}
R(x, (k_1, k_2)) &:= r \leftarrow {g_1}^{k_1} \cdot {g_2}^{k_2}, \\
S(e, (x_1, x_2), (k_1, k_2)) &:= (s_1, s_2) \leftarrow (k_1 + e \cdot x_1 \pmod{q}, k_2 + e \cdot x_2 \pmod{q}), \\
V(r, e, (s_1, s_2)) &:= \textbf{true} \text{ if and only if } (g_1^{s_1} \cdot g_2^{s_2} = y^e \cdot r), \\
S'(e, (s_1, s_2)) &:= r \leftarrow g_1^{s_1} \cdot g_2^{s_2} \cdot y^{-e}.
\end{aligned}$$

We leave it to the reader to verify that this protocol is complete and zero-knowledge, and to work out the knowledge extractor.

21.3.8. "Or" Proofs: Sometimes the statement about which we wish to execute a Sigma protocol is not as clear cut as the previous examples. For example, suppose we wish to show we know either a secret x or a secret y, without revealing which of the two secrets we know. This is a very common occurrence which arises in a number of advanced protocols, including the voting protocol we consider later in this chapter. It turns out that to show knowledge of one thing or another can be performed using an elegant protocol due to Cramer, Damgård and Schoenmakers.

First assume that there already exist Sigma protocols to prove knowledge of both secrets individually. We will combine these two Sigma protocols together into one protocol which proves the statement we require. The key idea is as follows: For the secret we know, we run the Sigma

protocol as normal, however, for the secret we do not know, we run the simulated Sigma protocol. These two protocols are then linked together by linking the commitments.

As a high-level example, suppose the Sigma protocol for proving knowledge of x is given by the set of algorithms

$$R_1(x, k_1), S_1(e_1, x, k_1), V_1(r_1, e_1, s_1), S_1'(e_1, s_1), E_1((r_1, e_1, s_1), (r_1, e_1', s_1')).$$

Similarly we let the Sigma protocol to prove knowledge of y be given by

$$R_2(y, k_2), S_2(e_2, y, k_2), V_2(r_2, e_2, s_2), S_2'(e_2, s_2), E_2((r_2, e_2, s_2), (r_2, e_2', s_2')).$$

We assume in what follows that the challenges e_1 and e_2 are bit strings of the same length. What is important is that they come from the same set $\mathbb{E}$, and can be combined in a one-time-pad-like manner.[4]

Now suppose we know x, but not y, then our algorithms for the combined proof become:

$$\begin{aligned}
R(x, k_1) &:= (r_1, r_2) \leftarrow \begin{cases} r_1 \leftarrow R_1(x, k_1) \\ e_2 \leftarrow \mathbb{E} \\ s_2 \leftarrow \mathbb{S}_2 \\ r_2 \leftarrow S_2'(e_2, s_2), \end{cases} \\
S(e, x, k_1) &:= (e_1, e_2, s_1, s_2) \leftarrow \begin{cases} e_1 \leftarrow e \oplus e_2 \\ s_1 \leftarrow S_1(e_1, x, k_1), \end{cases} \\
V((r_1, r_2), e, (e_1, e_2, s_1, s_2)) &:= \textbf{true} \text{ if and only if} \\
&\qquad e = e_1 \oplus e_2 \\
&\qquad \text{and } V_1(r_1, e_1, s_1) \\
&\qquad \text{and } V_2(r_2, e_2, s_2), \\
S'(e, (e_1, e_2, s_1, s_2)) &:= (r_1, r_2) \leftarrow (S_1'(e_1, s_1), S_2'(e_2, s_2).)
\end{aligned}$$

Note that the prover does not reveal the value of e_1, e_2 or s_2 until the response stage of the protocol. Also note that in the simulated protocol the correct distributions of e, e_1 and e_2 are such that $e = e_1 \oplus e_2$. The protocol for the case where we know y but not x follows by reversing the roles of e_1 and e_2, and r_1 and r_2 in the algorithms R, S and V. If the prover knows both x and y then they can execute either of the two possibilties. The completeness, soundness and zero-knowledge properties follow from the corresponding properties of the original Sigma protocols.

These "Or" proofs can be extended to an arbitrary number of disjunctions of statements in the obvious manner: Given n statements of which the prover only knows one secret,

- Simulate $n - 1$ statements using the simulations and challenges e_i
- Commit as usual to the known statement
- Generate a correct challenge for the known statement via

$$e = e_1 \oplus \cdots \oplus e_n.$$

Example 1: We now present a simple example which uses the Schnorr protocol as a building block. Suppose we wish to prove knowledge of either x_1 or x_2 such that $y_1 = g^{x_1}$ and $y_2 = g^{x_2}$, where g lies in a group G of prime order q. We assume that the prover knows x_i but not x_j where $i \neq j$.

The prover's commitment, (r_1, r_2), is computed by selecting e_j and k_i uniformly at random from ${\mathbb{F}_q}^*$ and s_j uniformly at random from G. They then compute $r_i \leftarrow g^{k_i}$ and $r_j \leftarrow g^{s_j} \cdot y_j^{-e_j}$.

[4] So for example we could use addition if they came from a finite field.

On receiving the challenge $e \in \mathbb{F}_q{}^*$ the prover computes

$$\begin{aligned} e_i &\leftarrow e - e_j \pmod q \\ s_i &\leftarrow k_i + e_i \cdot x_i \pmod q. \end{aligned}$$

Note that we have replaced $\oplus$ in computing the "challenge" e_i with addition modulo q; a moment's thought reveals that this is a better way to preserve the relative distributions in this example since arithmetic associated with the challenge is performed in $\mathbb{F}_q$. The prover then outputs (e_1, e_2, s_1, s_2). The verifier checks the proof by checking that $e = e_1 + e_2 \pmod q$ and $r_1 = g^{s_1} \cdot y_1^{-e_1}$ and $r_2 = g^{s_2} \cdot y_2^{-e_2}$.

Example 2: We end this section by giving a protocol which will be required when we discuss voting schemes. It is obtained by combining the protocol for proving knowledge of Pedersen commitments with "Or" proofs. Consider the earlier commitment scheme given by

$$B_a(x) \leftarrow h^x g^a,$$

where $G = \langle g \rangle$ is a finite abelian group of prime order q, h is an element of G whose discrete logarithm with respect to g is unknown, x is the value being committed to, and a is a random value. We are interested in the case where the value committed to is restricted to be either plus or minus one, i.e. $x \in \{-1, 1\}$. It will be important in our application for the person committing to prove that their commitment is from the set $\{-1, 1\}$ without revealing what the actual value of the committed value is. To do this we execute the following protocol.

- As well as publishing the commitment $B_a(x)$, Peggy also chooses random numbers d, r and w modulo q and then publishes α_1 and α_2 where

$$\alpha_1 \leftarrow \begin{cases} g^r \cdot (B_a(x) \cdot h)^{-d} & \text{if } x = 1 \\ g^w & \text{if } x = -1, \end{cases}$$

$$\alpha_2 \leftarrow \begin{cases} g^w & \text{if } x = 1 \\ g^r \cdot \left(B_a(x) \cdot h^{-1}\right)^{-d} & \text{if } x = -1. \end{cases}$$

- Victor now sends a random challenge e to Peggy.
- Peggy responds by setting

$$\begin{aligned} d' &\leftarrow e - d, \\ r' &\leftarrow w + a \cdot d'. \end{aligned}$$

  Then Peggy returns the values

$$(e_1, e_2, r_1, r_2) \leftarrow \begin{cases} (d, d', r, r') & \text{if } x = 1 \\ (d', d, r', r) & \text{if } x = -1. \end{cases}$$

- Victor then verifies that the following three equations hold;

$$\begin{aligned} e &= e_1 + e_2, \\ g^{r_1} &= \alpha_1 \cdot (B_a(x) \cdot h)^{e_1}, \\ g^{r_2} &= \alpha_2 \cdot (B_a(x) \cdot h^{-1})^{e_2}. \end{aligned}$$

To show that the above protocol works we need to show that

(1) If Peggy responds honestly then Victor will verify that the above three equations hold.
(2) If Peggy has not committed to plus or minus one then she will find it hard to produce a response to Victor's challenge which is correct.
(3) The protocol reveals no information to any party as to the exact value of Peggy's commitment, bar that it comes from the set $\{-1, 1\}$.

We leave the verification of these three points to the reader. Note that the above protocol can clearly be conducted in a non-interactive manner by defining $e = H(\alpha_1 \| \alpha_2 \| B_a(x))$.

21.4. An Electronic Voting System

In this section we describe an electronic voting system which utilizes some of the primitives we have been discussing in this chapter and in earlier chapters. In particular we make use of secret sharing schemes from Chapter 19, commitment schemes from Chapter 20, and zero-knowledge proofs from this chapter. The purpose is to show how basic cryptographic primitives can be combined into a complicated application giving real value. One can consider an electronic voting scheme to be a special form of secure multi-party computation, a topic which we shall return to in Chapter 22.

Our voting system will assume that we have m voters, and that there are n centres which perform the tallying. The use of a multitude of tallying centres is to allow voter anonymity and stop a few centres colluding to fix the vote. We shall assume that voters are only given a choice of two candidates, for example Democrat or Republican.

The voting system we shall describe will have the following seven properties.

(1) Only authorized voters will be able to vote.
(2) No one will be able to vote more than once.
(3) No stakeholder will be able to determine how someone else has voted.
(4) No one can duplicate someone else's vote.
(5) The final result will be correctly computed.
(6) All stakeholders will be able to verify that the result was computed correctly.
(7) The protocol will work even in the presence of some bad parties.

System Set-up: Each of the n tally centres has a public key encryption function E_i. We assume a finite abelian group G is fixed, of prime order q, and two elements $g, h \in G$ are selected for which no party (including the tally centres) knows the discrete logarithm $h = g^x$. Each voter has a public key signature algorithm, with which they sign all messages. This last point is to ensure only valid voters vote, and we will ignore this issue in what follows as it is orthogonal to the points we want to bring out.

Vote Casting: Each of the m voters picks a vote v_j from the set $\{-1, 1\}$. The voter picks a random blinding value $a_j \in \mathbb{Z}/q\mathbb{Z}$ and publishes their vote $B_j \leftarrow B_{a_j}(v_j)$, using the Pedersen commitment scheme. This vote is public to all participating parties, both tally centres and other voters. Along with the vote B_j the voter also publishes a non-interactive version of the protocol from Section 21.3.8 to show that the vote was indeed chosen from the set $\{-1, 1\}$. The vote and its proof are then digitally signed using the signing algorithm of the voter.

Vote Distribution: We now need to distribute the votes cast around the tally centres so that the final tally can be computed. To share the a_j and v_j around the tallying centres each voter employs Shamir secret sharing as follows: Each voter picks two random polynomials modulo q of degree $t < n$,

$$R_j(X) \leftarrow v_j + r_{1,j} \cdot X + \cdots + r_{t,j} \cdot X^t,$$
$$S_j(X) \leftarrow a_j + s_{1,j} \cdot X + \cdots + s_{t,j} \cdot X^t.$$

The voter computes

$$(u_{i,j}, w_{i,j}) = (R_j(i), S_j(i)) \text{ for } 1 \leq i \leq n.$$

The voter encrypts the pair $(u_{i,j}, w_{i,j})$ using the ith tally centre's encryption algorithm E_i. This encrypted share is sent to the relevant tally centre. The voter then publishes its commitment to the polynomial $R_j(X)$ by publicly posting $B_{l,j} \leftarrow B_{s_{l,j}}(r_{l,j})$ for $1 \leq l \leq t$, again using the earlier commitment scheme.

Consistency Check: Each centre i needs to check that the values of $(u_{i,j}, w_{i,j})$ it has received from voter j are consistent with the commitment made by the voter. This is done by verifying the following equation:

$$\begin{aligned} B_j \cdot \prod_{\ell=1}^{t} {B_{\ell,j}}^{i^\ell} &= B_{a_j}(v_j) \cdot \prod_{\ell=1}^{t} B_{s_{\ell,j}}(r_{\ell,j})^{i^\ell} \\ &= h^{v_j} \cdot g^{a_j} \cdot \prod_{\ell=1}^{t} (h^{r_{\ell,j}} \cdot g^{s_{\ell,j}})^{i^\ell} \\ &= h^{\left(v_j + \sum_{\ell=1}^{t} r_{\ell,j} \cdot i^\ell\right)} \cdot g^{\left(a_j + \sum_{\ell=1}^{t} s_{\ell,j} \cdot i^\ell\right)} \\ &= h^{u_{i,j}} g^{w_{i,j}}. \end{aligned}$$

Tally Counting: Tally centre i now computes and publicly posts its sum of the shares of the votes cast $T_i = \sum_{j=1}^{m} u_{i,j}$, plus it posts its sum of shares of the blinding factors $A_i = \sum_{j=1}^{m} w_{i,j}$. Every other party, both other centres and voters, can check that this has been done correctly by verifying that

$$\prod_{j=1}^{m} \left(B_j \cdot \prod_{\ell=1}^{t} {B_{\ell,j}}^{j^\ell} \right) = \prod_{j=1}^{m} h^{u_{i,j}} \cdot g^{w_{i,j}} = h^{T_i} \cdot g^{A_i}.$$

Any party can compute the final tally by taking t of the values T_i and interpolating them to reveal the final tally. This is because T_i is the evaluation at i of a polynomial which shares out the sum of the votes. To see this we have

$$\begin{aligned} T_i &= \sum_{j=1}^{m} u_{i,j} = \sum_{j=1}^{m} R_j(i) \\ &= \left(\sum_{j=1}^{m} v_j \right) + \left(\sum_{j=1}^{m} r_{1,j} \right) \cdot i + \cdots + \left(\sum_{j=1}^{m} r_{t,j} \right) \cdot i^t. \end{aligned}$$

If the final tally is negative then the majority of people voted -1, whilst if the final tally is positive then the majority of people voted $+1$. You should now convince yourself that the above protocol has the seven properties we said it would at the beginning.

Chapter Summary

- An interactive proof of knowledge leaks no information if the transcript could be simulated without the need for the secret information.
- Both interactive proofs and zero-knowledge proofs are very powerful constructs; they can be used to prove any statement in $\mathcal{PSPACE}$.
- Interactive proofs of knowledge can be turned into digital signature algorithms by replacing the challenge by the hash of the commitment concatenated with the message.
- Quite complicated protocols can then be built on top of our basic primitives of encryption, signatures, commitment and zero-knowledge proofs. As an example we gave an electronic voting protocol.

Further Reading

The book by Goldreich has more details on zero-knowledge proofs, whilst a good overview of this area is given in the first edition of Stinson's book. The voting scheme we describe is given in the paper of Cramer et al. from Eurocrypt.

R. Cramer, M. Franklin, B. Schoenmakers and M. Yung. *Multi-authority secret-ballot elections with linear work*. In Advances in Cryptology – Eurocrypt 1996, LNCS 1070, 72–83, Springer, 1996.

O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudo-randomness*. Springer, 1999.

D. Stinson. *Cryptography: Theory and Practice*. First Edition. CRC Press, 1995.