Chapter 6
Sigma Protocols and Efficient Zero-Knowledge[1]

A zero-knowledge proof is an interactive proof with the additional property that the verifier learns nothing beyond the correctness of the statement being proved. The theory of zero-knowledge proofs is beautiful and rich, and is a cornerstone of the foundations of cryptography. In the context of cryptographic protocols, zero-knowledge proofs can be used to enforce "good behavior" by having parties prove that they indeed followed the protocol correctly. These proofs must reveal nothing about the parties' private inputs, and as such must be zero knowledge. Zero-knowledge proofs are often considered an expensive (and somewhat naive) way of enforcing honest behavior, and those who view them in this way consider them to be not very useful when constructing efficient protocols. Although this is true for arbitrary zero-knowledge proofs of $\mathcal{NP}$ statements, there are many languages of interest for which there are extraordinarily efficient zero-knowledge proofs. Indeed, in many cases an efficient zero-knowledge proof is the best way to ensure that malicious parties do not cheat. We will see some good examples of this in Chapter 7 where efficient zero-knowledge proofs are used to achieve oblivious transfer with very low cost.

In this chapter, we assume familiarity with the notions of zero-knowledge and honest verifier zero-knowledge, and refer readers to [30, Chapter 4] for the necessary theoretical background, and for definitions and examples. As everywhere else in this book, here we are interested in achieving *efficiency.*

6.1 An Example

We begin by motivating the notion of a Σ-protocol through an example. Let p be a prime, q a prime divisor of $p-1$, and g an element of order q in $\mathbb{Z}_p^*$. Suppose that a prover P has chosen a random $w \leftarrow_R \mathbb{Z}_q$ and has published $h = g^w \bmod p$. A verifier V who receives (p, q, g, h) can efficiently

[1] Much of this chapter is based on a survey paper by Ivan Damgård [20]. We thank him for graciously allowing us to use his text.

verify that p, q are prime, and that g, h both have order q. Since there is only one subgroup of order q in Z_p^*, this automatically means that $h \in \langle g \rangle$ and thus there always exists a value w such that $h = g^w$ (this holds because in a group of prime order all elements apart from the identity are generators and so g is a generator). However, this does not necessarily mean that P *knows* w.

Protocol 6.1.1 by Schnorr provides a very efficient way for P to convince V that it knows the unique value $w \in \mathbb{Z}_q$ such that $h = g^w \bmod p$:

PROTOCOL 6.1.1 (Schnorr's Protocol for Discrete Log)

- **Common input:** The prover P and verifier V both have (p, q, g, h).
- **Private input:** P has a value $w \in \mathbb{Z}_q$ such that $h = g^w \bmod p$.
- **The protocol:**
  1. The prover P chooses a random $r \leftarrow_R \mathbb{Z}_q$ and sends $a = g^r \bmod p$ to V.
  2. V chooses a random *challenge* $e \leftarrow_R \{0,1\}^t$ and sends it to P, where t is fixed and it holds that $2^t < q$.
  3. P sends $z = r + ew \bmod q$ to V, which checks that $g^z = ah^e \bmod p$, that p, q are prime and that g, h have order q, and accepts if and only if this is the case.

Intuitively, this is a proof of knowledge because if some P^*, having sent a, can answer two *different* challenges e, e' correctly, then this means that it could produce z, z' such that $g^z = ah^e \bmod p$ and also $g^{z'} = ah^{e'} \bmod p$. Dividing one equation by the other, we have that $g^{z-z'} = h^{e-e'} \bmod p$. Now, by the assumption, $e - e' \neq 0 \bmod q$ (otherwise e and e' are not different challenges), and so it has a multiplicative inverse modulo q. Since g, h have order q, by raising both sides to this power, we get $h = g^{(z-z')(e-e')^{-1}} \bmod p$, and so $w = (z - z')(e - e')^{-1} \bmod q$. Observing that z, z', e, e' are all known to the prover, we have that the prover itself can compute w and thus *knows* the required value (except with probability 2^{-t}, which is the probability of answering correctly with a random guess). Thus, Protocol 6.1.1 is a *proof of knowledge*; we prove this formally below in Section 6.3.

In contrast, Protocol 6.1.1 is *not* known to be zero-knowledge. In order to see this, observe that in order for the problem of finding w to be non-trivial in the first place, q must be (exponentially) large. Furthermore, to achieve negligible error in a single run of the protocol, 2^t must be exponentially large too. In this case, standard rewinding techniques for zero-knowledge simulation will fail because it becomes too hard for a simulator to guess the value of e in advance. It is therefore not known if there exists some efficient malicious strategy that the verifier may follow which, perhaps after many executions of the protocol, enables it to obtain w (or learn more than is possible from (p, q, g, h) alone).

On the positive side, the protocol is *honest verifier zero-knowledge.* To simulate the view of an honest verifier V, simply choose a random $z \leftarrow_R \mathbb{Z}_q$

and $e \leftarrow_R \{0,1\}^t$, compute $a = g^z h^{-e} \bmod p$, and output (a, e, z). It is immediate that (a, e, z) has exactly the same probability distribution as a real conversation between the honest prover and the honest verifier. It is even possible to take any given value e and then produce a conversation where e occurs as the challenge: just choose z at random and compute the a that matches. In other words, the simulator does not have to insist on choosing e itself; it can take an e value as input. As we will see, this property turns out to be very useful.

One might argue that honest verifier zero-knowledge is not a very useful property, since it does not make sense in practice to assume that the verifier is honest. However, as we will see, protocols with this weaker property can often be used as building blocks in constructions that are indeed secure against active cheating, and are almost as efficient as the original protocol.

A final practical note: in a real-life scenario, it will often be the case where p and q are fixed for a long period, and so V can check primality once and for all, and will not have to do it in every execution of the protocol.

Working in general groups. From here on, we will consider general groups for proofs such as those above. The translation of Protocol 6.1.1 to this general language is as follows. The common input to the parties is a tuple $(\mathbb{G}, q, g, h)$ where $\mathbb{G}$ denotes a concise representation of a finite group of prime order q, and g and h are generators of $\mathbb{G}$. The proof then remains identical, with all operations now being in the group $\mathbb{G}$, and not necessarily operations mod$p$. In addition, the checks by V in the last step regarding the group are different. Specifically, V must be able to efficiently verify that $\mathbb{G}$ is indeed a group of prime order q, and that $g, h \in \mathbb{G}$ (note that in the concrete example above, V checks membership in $\mathbb{G}$ by checking that the element in question is of order q).

6.2 Definitions and Properties

Based on the Schnorr example, we will now define some abstract properties for a protocol that capture the essential properties of the example. It turns out that this abstraction holds for a wide class of protocols that are extremely useful when constructing efficient secure protocols.

Let R be a binary relation. That is, $R \subset \{0,1\}^* \times \{0,1\}^*$, where the only restriction is that if $(x, w) \in R$, then the length of w is at most $p(|x|)$, for some polynomial $p()$. For some $(x, w) \in R$, we may think of x as an instance of some computational problem, and w as the solution to that instance. We call w *a witness* for x.

For example, we could define a relation $\mathcal{R}_{\text{DL}}$ for the discrete log problem by

$$\mathcal{R}_{\text{DL}} = \{((\mathbb{G}, q, g, h), w) \mid h = g^w\},$$

where it is to be understood that $\mathbb{G}$ is of order q with generator g, that q is prime and that $w \in \mathbb{Z}_q$. In this case, R contains the entire set of discrete log problems of the type we considered above, together with their solutions.

We will be concerned with protocols of the form of Protocol 6.2.1.

PROTOCOL 6.2.1 (Protocol Template for Relation R)

- **Common input:** The prover P and verifier V both have x.
- **Private input:** P has a value w such that $(x, w) \in R$.
- **The protocol template:**

1. P sends V a message a.
2. V sends P a *random* t-bit string e.
3. P sends a reply z, and V decides to accept or reject based solely on the data it has seen; i.e., based only on the values (x, a, e, z).

Furthermore, we will assume throughout that both P and V are probabilistic polynomial time machines, and so P's only advantage over V is that it knows the witness w. We say that a transcript (a, e, z) is an accepting transcript for x if the protocol instructs V to accept based on the values (x, a, e, z). We now formally define the notion of a Σ-protocol (the name Σ-protocol comes from the form of the letter Σ that depicts a three-move interaction).

Definition 6.2.2 *A protocol π is a Σ-*protocol *for relation R if it is a three-round public-coin protocol of the form in Protocol 6.2.1 and the following requirements hold:*

- **Completeness:** *If P and V follow the protocol on input x and private input w to P where $(x, w) \in R$, then V always accepts.*
- **Special soundness:** *There exists a polynomial-time algorithm A that given any x and any pair of accepting transcripts $(a, e, z), (a, e', z')$ for x, where $e \neq e'$, outputs w such that $(x, w) \in R$.*
- **Special honest verifier zero knowledge:** *There exists a probabilistic polynomial-time simulator M, which on input x and e outputs a transcript of the form (a, e, z) with the same probability distribution as transcripts between the honest P and V on common input x. Formally, for every x and w such that $(x, w) \in R$ and every $e \in \{0,1\}^t$ it holds that*

$$\Big\{M(x,e)\Big\} \equiv \Big\{\langle P(x,w), V(x,e)\rangle\Big\}$$

where $M(x, e)$ denotes the output of simulator M upon input x and e, and $\langle P(x, w), V(x, e)\rangle$ denotes the output transcript of an execution between P and V, where P has input (x, w), V has input x, and V's random tape (determining its query) equals e.

The value t is called the challenge length.

It should be clear that Protocol 6.1.1 is a Σ-protocol. However, this protocol is not the exception to the rule. Rather, there are many such examples. This makes the general study of Σ-protocols important.

Until now, we have considered proofs of knowledge, and not proofs of membership. Observe that in the case of the discrete log there always exists a w for which $h = g^w$; the question is just whether or not P knows w. However, Σ-protocols often also arise in the setting of proofs of membership. Define L_R to be the set of inputs x for which there exists a w such that $(x, w) \in L_R$. Then the special soundness property implies that a Σ-protocol for R is always an interactive proof system for L_R with error probability 2^{-t}. This is due to the fact that if the special soundness algorithm A must be able to output w given any two accepting transcripts, then there must only be a *single* accepting transcript whenever $x \notin L_R$. Formally,

Proposition 6.2.3 *Let π be a Σ-protocol for a relation R with challenge length t. Then, π is an interactive proof of membership for L_R with soundness error 2^{-t}.*

Proof. Let $x \notin L_R$. We show that no P^* can convince V to accept with probability greater than 2^{-t}, even if P^* is computationally unbounded. Assume by contradiction that P^* can convince V with probability greater than 2^{-t}. This implies that there exists a first message a from P^* and at least two queries e, e' from V that result in accepting transcripts. This is the case because if for every a there exists at most one query e that results in an accepting transcript, then P^* would convince V with probability at most 2^{-t}; i.e., P^* would convince V only if V chose the single query e that P^* can answer, and since $e \leftarrow_R \{0,1\}^t$ this happens with probability only 2^{-t}. Observe now that the special soundness property requires that A output a witness w such that $(x, w) \in R$ (implying that $x \in L_R$) when given *any* pair of accepting transcripts $(a, e, z), (a, e', z')$ with $e \neq e'$. Thus, we conclude that whenever P^* can convince V with probability greater than 2^{-t} it holds that $x \in L_R$, in contradiction to the assumption. ■

Before proceeding, we give another example of a useful Σ-protocol that is also a proof of membership (i.e., unlike the case of the discrete log, here the fact that the input is in the language is non-trivial to verify). Specifically, we consider the case of proving that a tuple $(\mathbb{G}, q, g, h, u, v)$ is of the Diffie-Hellman form, that is, that there exists a w such that $u = g^w$ and $v = h^w$. By the decisional Diffie-Hellman assumption, the case where $u = g^w$ and $v = h^w$ for some w is computationally indistinguishable from the case where $u = g^w$ and $h = g^{w'}$ for $w \neq w'$. Thus, a proof of this fact is non-trivial. We remark that proofs of this type come up in many settings. For just one example, this exact proof is utilized by Protocol 7.4.1 for securely computing oblivious transfer in the malicious setting; see Section 7.4. Formally, Protocol 6.2.4 below is a proof system for the relation

$$\mathcal{R}_{\text{DH}} = \left\{ ((\mathbb{G}, q, g, h, u, v), w) \mid g, h \in \mathbb{G} \ \& \ u = g^w \ \& \ v = h^w \right\}.$$

PROTOCOL 6.2.4 (Σ Protocol for Diffie-Hellman Tuples)

- **Common input:** The prover P and verifier V both have $(\mathbb{G}, q, g, h, u, v)$.
- **Private input:** P has a value w such that $u = g^w$ and $v = h^w$.
- **The protocol:**
  1. The prover P chooses a random $r \leftarrow_R \mathbb{Z}_q$ and computes $a = g^r$ and $b = h^r$. It then sends (a, b) to V.
  2. V chooses a random challenge $e \leftarrow_R \{0,1\}^t$ where $2^t < q$ and sends it to P.
  3. P sends $z = r + ew \bmod q$ to V, which checks that $g^z = au^e$ and $h^z = bv^e$, that $\mathbb{G}$ is a group of order q with generators g and h, and that $u, v \in \mathbb{G}$, and accepts if and only if this is the case.

Claim 6.2.5 *Protocol 6.2.4 is a Σ-protocol for the relation $\mathcal{R}_{\rm DH}$.*

Proof. In order to see that completeness holds, observe that when P runs the protocol honestly we have

$$g^z = g^{r+ew} = g^r \cdot g^{we} = g^r \cdot (g^w)^e = a \cdot u^e$$

where the last equality is due to the fact that $a = g^r$ and $u = g^w$. A similar calculation shows that $h^z = bv^e$. Regarding special soundness, let $((a,b), e, z)$ and $((a,b), e', z')$ be two accepting transcripts. We have that $g^z = au^e$ and $g^{z'} = au^{e'}$, as well as $h^z = bv^e$ and $h^{z'} = bv^{e'}$. Dividing both pairs of equations we have that $g^{z-z'} = u^{e-e'}$ and $h^{z-z'} = v^{e-e'}$. Continuing as in Schnorr's protocol, we conclude that $u = g^{(z-z')/(e-e')}$ and $v = h^{(z-z')/(e-e')}$. The machine A can therefore set $w = \frac{z-z'}{e-e'}$ and it holds that $((\mathbb{G}, q, g, h, u, v), w) \in \mathcal{R}_{\rm DH}$, as required.

It remains to demonstrate special honest verifier zero knowledge. However, this is also almost identical to the simulator in Protocol 6.1.1, and is therefore omitted. ■

Properties of Σ-protocols. We conclude this section with two important, yet easily verified properties of Σ-protocols. We first consider parallel repetition, where the parties run the same protocol multiple times with the same input, and V accepts if and only if it accepts in all repetitions. We stress that the parties use independent randomness for generating their messages in every execution.

Lemma 6.2.6 *The properties of Σ-protocols are invariant under parallel repetition. That is, the parallel repetition ℓ times of a Σ-protocol for R with challenge length t yields a new Σ-protocol for R with challenge length $\ell \cdot t$.*

Combining this with Proposition 6.2.3 we have that parallel repetition reduces the error exponentially fast. It is worthwhile noting that if there are two accepting transcripts of the parallel protocol then this actually means that there are ℓ pairs of accepting transcripts. For this reason, the soundness error is reduced to $2^{-\ell \cdot t}$ as expected.

Next we show that the challenge can be set to any arbitrary length.

Lemma 6.2.7 *If there exists a Σ-protocol π for R with challenge length t, then there exists a Σ-protocol π' for R with challenge length t', for any t'.*

Proof. Let t be the challenge length for the given protocol π. Then a Σ-protocol with challenge length t' shorter than t can be constructed as follows. P sends the first message a as in π. V then sends a random t'-bit string e. Finally, P appends $t-t'$ 0s to e, calls the result e', and computes the answer z to e' as in π. The verifier V checks z as in π as if the challenge were e'. Special soundness trivially still holds. In addition, special zero-knowledge holds because the simulator M is required to work for *every* challenge e, including those the conclude with $t-t'$ 0s.

Regarding longer challenge lengths $t' > t$; this can be achieved by first running π in parallel j times, for j such that $jt \geq t'$ (as in Lemma 6.2.6), and then adjusting jt down to t' as above in the case where $jt > t'$. ■

6.3 Proofs of Knowledge

In this section we prove that any Σ-protocol with challenge length t is a proof of knowledge with knowledge error 2^{-t}. We begin by reviewing the notion of a proof of knowledge, as defined in [7]. It should be noted that it is not at all straightforward to define this notion. Specifically, what does it mean for a machine to "know" or "not know" something? The aim of a proof of knowledge is to enable a prover to convince a verifier that it knows a witness for some statement. Thus, it must be the case that only a machine that "knows" a witness should be able to convince the verifier. The definition below formalizes this by essentially saying that a machine knows a witness if it can be used to efficiently compute it. Stated differently, if a witness can be efficiently computed given access to P^*, then this means that P^* knows the witness. Indeed, P^* can run the efficient computation on itself and thereby explicitly obtain the witness. We refer to [30, Chapter 4] for more discussion, and proceed directly to the definition. (Note that the definition here is as in [7] and differs from that presented in [30]. Specifically, we consider probabilistic provers as in [7] and not only deterministic ones as considered in [30]. See [8] for a discussion regarding the difference between these definitions.)

Definition 6.3.1 *Let $\kappa : \{0,1\}^* \to [0,1]$ be a function. A protocol (P,V) is a* proof of knowledge *for the relation R with* knowledge error *κ if the following properties are satisfied:*

Completeness: *If P and V follow the protocol on input x and private input w to P where $(x,w) \in R$, then V always accepts.*

Knowledge soundness (or validity): *There exists a constant $c > 0$ and a probabilistic oracle machine K, called the* knowledge extractor*, such that for every interactive prover function P^* and every $x \in L_R$, the machine K satisfies the following condition. Let $\epsilon(x)$ be the probability that V accepts on input x after interacting with P^*. If $\epsilon(x) > \kappa(x)$, then upon input x and oracle access to P^*, the machine K outputs a string w such that $(x,w) \in R$ within an expected number of steps bounded by*

$$\frac{|x|^c}{\epsilon(x) - \kappa(x)}.$$

One can think of the error κ as the probability that one can convince the verifier without knowing a valid w, and the ability to convince the verifier with higher probability means that the prover knows w. Furthermore, the higher the probability that P^* convinces V, the more efficient it is to compute w.

As we have mentioned, we prove here that any Σ-protocol with challenge length t is a proof of knowledge with knowledge error 2^{-t}. Intuitively Σ-protocols are proofs of knowledge because this is essentially what the special soundness property states. Specifically, as we have seen, if P can convince V with probability greater than 2^{-t} then there must be two accepting transcripts, in which case it is possible to apply the extraction machine guaranteed to exist by special soundness. The proof that this indeed holds is however more involved because the witness extractor K has to first *find* two accepting transcripts for such a prover, whereas the special soundness machine only needs to work when somehow magically given such a pair of transcripts. We now prove this theorem.

Theorem 6.3.2 *Let π be a Σ-protocol for a relation R with challenge length t. Then π is a proof of knowledge with knowledge error 2^{-t}.*

Proof. Completeness (or non-triviality) is clear by definition. We know prove the *validity* property; i.e., the existence of an extractor K as described above. Let H be the 0/1-matrix with a row for each possible set of random choices ρ by P^*, and one column for each possible challenge value e. An entry $H_{\rho,e}$ equals 1 if V accepts with this random choice and challenge, and 0 otherwise. Using P^* as a black box while setting its internal random coins to be random and choosing a random challenge, we can probe a random entry in H. Furthermore, by rewinding P^* and reusing the same internal random coins as before, we can probe a random entry in the same row. In this light, our goal is to find two 1s in the same row. Using special soundness the resulting two transcripts provide us with sufficient information to efficiently compute a witness w for x.

Let P^* be a prover that convinces V upon input x with probability $\epsilon(x)$, and assume that $\epsilon(x) > 2^{-t}$. This implies that $\epsilon := \epsilon(x)$ equals the fraction of 1-entries in H. However, it is important to note that this gives *no guarantees* about the distribution of 1s in a given row. For instance, there may be some rows with many 1s and others with few 1s (a row with a single 1 cannot be used to find w because it does not give two accepting transcripts). Despite the above, we can make the following observation about the distribution of 1s in H. Define a row to be heavy if it contains a fraction of at least $\epsilon/2$ ones (there are 2^t entries in a row, and so this means that there are $\epsilon \cdot 2^{t-1}$ 1s in a heavy row). By a simple counting argument, it holds that more than half of the 1s are located in heavy rows. In order to see this, let H' be the sub-matrix of H consisting of all rows that are not heavy, and denote by h' the total number of entries in H', and by h the total number of entries in H. By the assumption, the number of 1s in H is $h \cdot \epsilon$, and the number of 1s in H' is smaller than $h' \cdot \epsilon/2$. This implies that the number of 1s in heavy rows is greater than

$$h \cdot \epsilon - h' \cdot \frac{\epsilon}{2} \geq h \cdot \epsilon - h \cdot \frac{\epsilon}{2} = \frac{h \cdot \epsilon}{2},$$

demonstrating that half of the 1s are indeed in heavy rows.

We first show how the extractor works under the assumption that

$$\epsilon \geq 2^{-t+2},$$

so that a heavy row contains at least two 1s (recall that a heavy row has at least $\epsilon/2$ 1s, and so if ϵ is at least 2^{-t+2}, there are at least $2^{-t+2}/2 \cdot 2^t = 2$ 1s in each such row). In this case, we will show that we can find two 1s in the same row in expected time $O(1/\epsilon)$. This will be more than sufficient since $1/\epsilon$ is less than required by the definition, namely $1/(\epsilon - 2^{-t})$.

Our approach will be to first repeatedly probe H at random, until we find a 1 entry, a "first hit". This happens after an expected number of $1/\epsilon$ tries, because the fraction of 1s in H is exactly ϵ. By the observation above, with probability greater than 1/2, the first hit lies in a heavy row. Now, *if* it does (but note that we cannot check if it does), and if we continue probing at random along this row, the probability of finding another 1 in one attempt equals

$$\frac{\epsilon/2 \cdot 2^t - 1}{2^t}$$

because there are $\frac{\epsilon}{2} \cdot 2^t$ 1s in this row (by the assumption that it is heavy) and we need to find a different 1. This implies that the expected number T of tries to find the second hit satisfies

$$T = \frac{2^t}{\epsilon/2 \cdot 2^t - 1} = \frac{2^t}{\epsilon/2 \cdot (2^t - 2/\epsilon)} = \frac{2}{\epsilon} \cdot \frac{2^t}{2^t - 2/\epsilon} \leq \frac{2}{\epsilon} \cdot \frac{2^t}{2^t - 2^{t-1}} = \frac{4}{\epsilon}$$

tries, where the inequality follows from the assumption that $\epsilon \geq 2^{-t+2}$ above.

We therefore conclude that if the extractor's first hit was in a heavy row, then it finds two different accepting transcripts, as required for obtaining the witness, within an expected $O(1/\epsilon)$ tries. However, it may not be the case that the extractor's first hit was in a heavy row, and in such a case, we may spend too much time finding another 1 (if it exists at all). To remedy this, we include an "emergency break", resulting in the following algorithm (which is still not the final extractor, as we will see below):

1. As above, probe random entries in H until the first 1 is found (the first hit). The row in which this is found is called the current row.
2. Next, start the following two processes in *parallel*, and stop when either one stops:
   - **Process Pr_1:** Probe random entries in the current row, until another 1 entry is found (the second hit).
   - **Process Pr_2:** Repeatedly flip a coin that comes out heads with probability ϵ/d, for some *constant* d (we show how to choose d below) until you get heads. This can be achieved by probing a random entry in H and choosing a random number out of $1, 2, ..., d$. Then, output heads if both the entry in H is a 1 and the number chosen is 1.

We now analyze the expected running time of the above algorithm and its probability of success. Regarding the running time, observe that whenever one of the processes stops the entire algorithm halts. Thus, it suffices to analyze the expected running time of Pr_2 and this provides an upper bound on the expected running time of the entire algorithm. Now, since Pr_2 halts when the first heads appears, we have that its expected running time is d/e, which for a constant d equals $O(1/\epsilon)$ as required. We now show that for an appropriate choice of the constant d, the algorithm succeeds in extracting a witness with probability at least 1/8. Observe that the algorithm only succeeds in extracting a witness if Pr_1 finishes first. Therefore, we have to choose d so that Pr_1 has enough time to finish before Pr_2, if indeed the first hit is in a heavy row.

The probability that Pr_2 finishes after exactly k attempts is $\epsilon/d \cdot (1 - \epsilon/d)^{k-1}$. Using the crude estimate $(1-\epsilon/d)^{k-1} \leq 1$, we have that the probability that Pr_2 finishes *within* k attempts is less than or equal to $k\epsilon/d$. Thus, by setting $k = 8/\epsilon$ and $d = 16$ we have that the probability that Pr_2 finishes within $8/\epsilon$ attempts is less than or equal to

$$\frac{k\epsilon}{d} = \frac{\frac{8}{\epsilon} \cdot \epsilon}{16} = \frac{1}{2}.$$

We conclude that setting $d = 16$, it holds that Pr_2 finishes after more than $8/\epsilon$ tries with probability *at least* 1/2.

We are now ready to show that the algorithm succeeds in extracting a witness with probability at least 1/8. In order to see this, first recall that *if* the first hit is in a heavy row, then the expected number of steps of Pr_1 is at most $T = \epsilon/4$ and so by Markov's inequality, the probability that Pr_1 takes $2T \leq 8/\epsilon$ or more steps is upper-bounded by $1/2$. Stated differently, if the first hit is in a heavy row then with probability at least $1/2$ process Pr_1 concludes in less than $8/\epsilon$ steps. Since the random variables determining the stopping condition of Pr_1 and Pr_2 are independent, we have that if the first hit is in a heavy row then Pr_1 finishes before Pr_2 with probability at least $1/2 \cdot 1/2 = 1/4$. Next, recall that the first hit is in a heavy row with probability at least $1/2$. We have that the probability that the algorithm succeeds equals at least the probability that the first hit is a heavy row times the probability that Pr_1 finishes before Pr_2, *conditioned* on the first hit being a heavy row. Thus, the algorithm succeeds with probability $1/2 \cdot 1/4 = 1/8$, as required.

We now describe the actual knowledge extractor (however, still only for the case where $\epsilon \geq 2^{-t+2}$). The knowledge extractor K works by repeating the above algorithm until it succeeds. Since the expected number of repetitions is constant (at most eight), we have that K succeeds in finding a witness in an expected number of steps that is bound by $O(1/\epsilon)$, as desired.

It still remains to consider the case where $2^{-t} < \epsilon < 2^{-t+2}$; recall that above we assumed that $\epsilon \geq 2^{-t+2}$ but the extractor K must work whenever $\epsilon \geq 2^{-t}$. We treat this case by a separate algorithm, using the fact that when ϵ is so small, we actually have enough time to probe an entire row. The knowledge extractor K runs this algorithm in parallel with the above one.

Define δ to be such that $\epsilon = (1+\delta)2^{-t}$. Note that for the values of ϵ that we consider here it holds that $0 < \delta < 3$. Let R be the number of rows in H. Then we have that the number of 1s in H is at least $(1+\delta) \cdot R$, where the total number of entries in H equals $R \cdot 2^t$. Since there are only R rows, it follows that at most $R-1$ of the $(1+\delta)R$ ones can be alone in a row, and thus at least δR of them must be in rows with at least two 1s. We call such a row semi-heavy. The algorithm here works as follows:

1. Probe random entries until a 1 is found; call this row the current row.
2. Search the entire current row for another 1 entry. If no such entry is found, then go back to Step 1.

It is clear that the algorithm successfully finds a witness. However, the question is whether it does so in time $O(1/\epsilon - 2^{-t})$. In order to analyze this, note that the fraction of 1s in semi-heavy rows is $\delta/(1+\delta)$ among all 1s (because the fraction of 1s overall is $(1+\delta)$ and a δ fraction of these are in semi-heavy rows). In addition, the fraction of 1s in semi-heavy rows is at least $\delta/2^t$ among all entries.

Now, the expected number of probes to find a 1 is

$$\frac{1}{\epsilon} = \frac{2^t}{(1+\delta)}. \tag{6.1}$$

Furthermore, by the fact that the fraction of 1s in semi-heavy rows among all entries is $\delta/2^t$, the expected number of probes to find a 1 in a semi-heavy row is $2^t/\delta$. We therefore expect to find a 1 in a semi-heavy row after finding $(1+\delta)/\delta$ 1s. This implies that the expected number of times that the algorithm carries out Steps 1 and 2 equals $(1+\delta)/\delta$. In each such repetition, the expected number of probes in Step 1 equals $2^t/(1+\delta)$; see (6.1). Then, once a 1 is found, the number of probes in Step 2 is exactly 2^t. We therefore have that the expected cost is

$$\frac{1+\delta}{\delta} \cdot \left(\frac{2^t}{1+\delta} + 2^t \right) = 2^t \cdot \left(\frac{1}{\delta} + \frac{1+\delta}{\delta} \right) = 2^t \cdot \frac{2+\delta}{\delta} < 5 \cdot \frac{2^t}{\delta}.$$

However,

$$\frac{1}{\epsilon - 2^{-t}} = \frac{1}{(1+\delta)2^{-t} - 2^{-t}} = \frac{2^t}{\delta},$$

proving that the algorithm runs in time $O(1/(\epsilon - 2^{-t}))$, as required. This completes the proof that any Σ-protocol with challenge length t is a proof of knowledge with knowledge error 2^{-t}. ■

6.4 Proving Compound Statements

In this section, we show how basic Σ-protocols can be used to prove compound statements. This is a very useful tool in cryptographic protocols and is a good demonstration of the usefulness of Σ-protocols. It is easy to prove the AND of two statements: simply have the prover prove both in parallel with the verify sending a single challenge e for both proofs. It is easy to see that the result is a Σ-protocol for a compound relation comprised of the AND of two statements. We leave the proof of this as an exercise.

In contrast, proving the OR of two statements is more involved. Specifically, the problem we wish to solve is as follows. A prover and verifier P and V hold a pair (x_0, x_1) for their common input. The prover P wishes to prove to V that it knows a witness w such that either $(x_0, w) \in R$ or $(x_1, w) \in R$, *without revealing which is the case.* We use this methodology in Section 7.5 in an oblivious transfer protocol where one party has to prove to another that one of two tuples is a Diffie-Hellman tuple, without revealing which. We remark that similar ideas can be used to extend the construction below to prove that k out of n statements are true, again without revealing which; see [19]. In addition, the construction below remains unchanged when the statements are for different relations R_0 and R_1 and the goal is for P to prove that it knows a witness w such that either $(x_0, w) \in R_0$ or $(x_1, w) \in R_1$. We present the case of a single relation for the sake of clarity.

Assume that we are given a Σ-protocol π for a relation R. Assume also that the pair (x_0, x_1) is common input to P and V, and that w is the private input of P, where either $(x_0, w) \in R$ or $(x_1, w) \in R$. Let b be such that $(x_b, w) \in R$. Roughly speaking, the idea is to have the prover complete two instances of π, one with input x_0 and the other with input x_1. Of course, the problem is that the prover can only do this for x_b, because it only knows a witness for this statement (indeed, the other statement x_{1-b} may not even be in the language L_R). Thus, the prover "fakes" the other statement by running the simulator M. This does not seem to solve the problem because in order to simulate, M needs to be able to determine e_{1-b} by itself, or at least to know e_{1-b} before generating a_{1-b}. Thus, the verifier V will know in which execution its challenge is used, and in which execution the prover set the challenge. This problem is overcome in the following fascinating way. The prover runs M to obtain $(a_{1-b}, e_{1-b}, z_{1-b})$. The prover P then sends V the first message a_{1-b} from the simulated transcript along with the first message a_b of the real execution. After receiving a challenge s from V, the prover P sets the challenge for the real execution to be $e_b = s \oplus e_{1-b}$ and completes both executions using challenges e_0 and e_1, respectively. (Observe that this means that P sends z_{1-b} from the simulated transcript in the simulated execution, as required.) Now, soundness still holds because a_{1-b} *binds* P to the challenge e_{1-b} (by the special soundness of π, if P does not know w such that $(x_{1-b}, w) \in R$ then it cannot complete the proof with any $e'_{1-b} \neq e_{1-b}$). Thus, the first message a_{1-b} together with the challenge s from the verifier actually defines a *unique* and *random* challenge $e_b = s \oplus e_{1-b}$ for the prover in the real execution. Thus, the prover must know w such that $(x_b, w) \in R$ in order to complete this execution. Interestingly, the verifier cannot distinguish which of the executions is real and which is simulated, because the transcript generated by M looks the same as a real one, and because $s \oplus e_{1-b}$ is distributed identically to $s \oplus e_b$. We therefore have that P can only complete the proof if it has a witness to one of the statements, and yet V cannot know which witness P has. Protocol 6.4.1 contains the full details of the OR protocol.

PROTOCOL 6.4.1 (OR Protocol for Relation R Based on π)

- **Common input:** The prover P and verifier V both have a pair (x_0, x_1).
- **Private input:** P has a value w and a bit b such that $(x_b, w) \in R$.
- **The protocol:**
  1. P computes the first message a_b in π, using (x_b, w) as input.
     P chooses e_{1-b} at random and runs the simulator M on input (x_{1-b}, e_{1-b}); let $(a_{1-b}, e_{1-b}, z_{1-b})$ be the output of M.
     P sends (a_0, a_1) to V.
  2. V chooses a random t-bit string s and sends it to P.
  3. P sets $e_b = s \oplus e_{1-b}$ and computes the answer z_b in π to challenge e_b using (x_b, a_b, e_b, w) as input. P sends (e_0, z_0, e_1, z_1) to V.
  4. V checks that $e_0 \oplus e_1 = s$ and that both transcripts (a_0, e_0, z_0) and (a_1, e_1, z_1) are accepting in π, on inputs x_0 and x_1, respectively.

Let $R_{\text{OR}} = \{((x_0, x_1), w) \mid (x_0, w) \in R \text{ or } (x_1, w) \in R\}$. Then we have:

Theorem 6.4.2 *Protocol 6.4.1 is a Σ-protocol for the relation R_{OR}. Moreover, for any verifier V^*, the probability distribution over transcripts between P and V^*, where w is such that $(x_b, w) \in R$, is independent of b.*

Proof. It is clear that the protocol is of the right form. To verify special soundness, let there be two accepting transcripts $(a_0, a_1, s, e_0, e_1, z_0, z_1)$ and $(a_0, a_1, s', e_0', e_1', z_0', z_1')$ where $s \neq s'$. Since the transcripts are accepting it holds that $e_0 \oplus e_1 = s$ and $e_0' \oplus e_1' = s'$. Since $s \neq s'$ this implies that for some $c \in \{0, 1\}$ it must hold that $e_c \neq e_c'$. By the special soundness of π this implies that from (a_c, e_c, z_c) and (a_c, e_c', z_c') it is possible to efficiently compute w such that $(x_c, w) \in R$, implying in turn that $((x_0, x_1), w) \in R_{\text{OR}}$.

Special honest verifier zero-knowledge is immediate: given s, choose e_0 and e_1 at random subject to $s = e_0 \oplus e_1$ and run M twice, once with input (x_0, e_0) and once with input (x_1, e_1).

It remains to prove that the probability distribution over transcripts is independent of b. Let V^* be an arbitrary verifier. Then, observe that the distribution over transcripts between P and V^* can be specified as follows. A transcript is of the form $(a_0, a_1, s, e_0, e_1, z_0, z_1)$, where a_0, a_1 are distributed as an honest prover in π would choose them (this is immediate for a_b and holds for a_{1-b} by the perfect honest verifier zero-knowledge of π). Then s has whatever distribution V^* outputs, given (x_0, x_1, a_0, a_1), and e_0, e_1 are random subject to $s = e_0 \oplus e_1$. Finally, z_0 and z_1 both have whatever distribution the honest prover in π outputs upon the respective transcript prefixes (x_0, a_0, e_0) and (x_1, a_1, e_1). As above, this is immediate for z_b and holds for z_{1-b} by the perfect honest verifier zero-knowledge of π. We therefore conclude that the distribution is independent of b. ■

By the last claim in this theorem, Protocol 6.4.1 is what is known as *witness indistinguishable* [27]. There are several different values of w that a prover may know that would enable it to complete the protocol successfully. However, there is no way that a verifier (even a malicious one) can know which of the possible witnesses the prover knows. This is a first sign that it is possible to obtain security properties that hold for arbitrary verifiers, even when starting from a protocol that is only honest verifier zero knowledge.

6.5 Zero-Knowledge from Σ-Protocols

In this section, we show how to construct efficient zero-knowledge protocols from any Σ-protocol. We remark that this can be done in a number of ways. We will first present a construction using any perfectly-hiding commitment scheme. This construction achieves zero knowledge, but is *not* a proof of knowledge. We will then show what needs to be modified so that the result is a zero-knowledge *proof of knowledge.*