On Σ-protocols

Ivan Damgård

CPT 2004

1 An example

Let p be a prime, q a prime divisor in $p-1$, and g an element of order q in Z_p^*. Suppose a prover P has chosen w in Z_q at random and has published $h = g^w \bmod p$. A verifier V who gets p, q, g, h can check that p, q are prime, and that g, h have order q. Since there is only one subgroup of order q in Z_p^*, this automatically means that $h \in < g >$, i.e. there exists w such that $h = g^w$. But this does not necessarily mean that P knows such a w.

The following protocol suggested by Schnorr gives a very efficient way to convince V about this:

1. P chooses r at random in Z_q and sends $a = g^r \bmod p$ to V.
2. V chooses a *challenge* e at random in Z_{2^t} and sends it to P. Here, t is fixed such that $2^t < q$.
3. P sends $z = r + ew \bmod q$ to V, who checks that $g^z = ah^e \bmod p$, that p, q are prime and that g, h have order q, and accepts iff this is the case.

We define proofs of knowledge later on in this note, but intuitively, if some P^*, having sent a, could answer two *different* challenges e, e' correctly, this would mean that he 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 get that $g^{z-z'} = h^{e-e'} \bmod p$. Now by assumption $e - e' \neq 0 \bmod q$, and so it has a multiplicative inverse modulo q. Since g, h have order q, by raising both sides to this power, we get $h = g^{(z-z')(e-e')^{-1}} \bmod p$, in other words $w = (z-z')(e-e')^{-1} \bmod q$. So loosely speaking, a cheating prover who does not know w can only be able to answer at most one challenge value correctly, since otherwise the argument we just gave would imply that he was in fact able to compute w after all. Thus the error probability for this proof is $1/2^t$. This is formalized later on, using the standard definition of proofs of knowledge by Bellare and Goldreich.

On the other hand the protocol is NOT known to be zero-knowledge: 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 will fail to apply, 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 "steal" w.

The protocol is *honest verifier zero-knowledge*, however: To simulate, simply choose at random $z \in Z_p^*$ and $e \in Z_q$ and compute $a = g^z h^{-e} \bmod p$. Then clearly (a, e, z) has exactly the same probability distribution as real conversations 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 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.

One might argue that honest verifier zero-knowledge is not a useful property, since it does not make sense in practice to assume that V is honest. The point, however, is that protocols with this weaker property can often be used as building blocks in constructions that are indeed secure against active cheating, as we shall see.

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

2 Definitions

Based on the Schnorr example, we will now define some abstract properties for a protocol that capture the essential properties of the example.

First, let R be a binary relation, i.e., R is a subset of $\{0,1\}^* \times \{0,1\}^*$, 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 DL by

$$DL = \{(x, w)|\ x = (p, q, g, h), ord(g) = ord(h) = q, h = g^w\},$$

where it is to be understood that p and q are prime, $g, h \in Z_p^*$, and $w \in Z_q$. So in this case, R contains the entire set of discrete log problems of the type we looked at above, together with their solutions.

We will be concerned with protocols of the following form, where x is common input to P, V and a w such that $(x, w) \in R$ is private input to P:

1. P sends a message a.
2. V sends a random t-bit string e.
3. P sends a reply z, and V decides to accept or reject based on the data he has seen, i.e. x, a, e, z.

We will assume throughout that both P, V are probabilistic polynomial time machines, so P's only advantage over V is that he knows w.

Definition 1. *A protocol $\mathcal{P}$ is said to be a Σ-protocol for relation R if:*

- *$\mathcal{P}$ is of the above 3-move form, and if P, V follow the protocol, the verifier always accepts.*
- *From any x and any pair of accepting conversations on input x, $(a, e, z), (a, e', z')$ where $e \neq e'$, one can efficiently compute w such that $(x, w) \in R$. This is sometimes called the* special soundness property*.*
- *There exists a polynomial-time simulator M, which on input x and a random e outputs an accepting conversation of the form (a, e, z), with the same probability distribution as conversations between the honest P, V on input x. This is sometimes called* special honest-verifier zero-knowledge*.*

It should be clear that the above example by Schnorr is a special case of this definition. The point, however, is that many such examples exist. And so any protocol we can build using Σ-protocols as subroutines will automatically work no matter which example protocol we use.

Define L_R to be the set of x's for which there exist w such that $(x, w) \in L$. 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 also the case for our example relation DL. Here, however, this is not so interesting because the verifier could very easily check membership in L_R on his own. However, even if deciding membership in L_R is easy, it still may make sense for P to demonstrate that he knows a w corresponding to a given x.

Here are some easily verified properties:

Lemma 1. *The properties of Σ-protocols are invariant under parallel composition, for instance repeating a Σ-protocol for R twice in parallel produces a new Σ-protocol for R with challenge length $2t$.*

Lemma 2. *If a Σ-protocol for R exists, then for any t, there exists a Σ-protocol for R with challenge length t.*

Proof. Let t' be the challenge length for the given protocol $\mathcal{P}$. Then any challenge length t shorter than t' can be implemented as follows: P sends the first message a as in $\mathcal{P}$. V sends a random t-bit string e. P appends $t' - t$ zeros to e, calls the result e' and computes the answer z to e' as in $\mathcal{P}$. V checks z as in $\mathcal{P}$, as if the challenge was e'.

Any challenge length $t > t'$ can be implemented by first repeating the given protocol in parallel j times, such that $jt' \geq t$, and then possibly adjusting down to t as above.

3 Proofs of Knowledge

Here, we give the formal definition of proofs of knowledge that was proposed by Bellare and Goldreich [1], and has become the standard one.

Definition 2. *Let $\kappa()$ be a function from bit strings to the interval $[0..1]$. The protocol (P, V) is said to be a proof of knowledge for the relation R with knowledge error κ, if the following are satisfied:*

Completeness *On common input x, if the honest prover P gets as private input w such that $(x, w) \in R$, then the verifier V always accepts.*

Knowledge soundness *There exists a probabilistic algorithm M called the knowledge extractor. This M gets input x and rewindable black-box access to the prover and attempts to compute w such that $(x, w) \in R$. We require that the following holds: For any prover P^*, let $\epsilon(x)$ be the probability that V accepts on input x. There exists a constant c such that whenever $\epsilon(x) > \kappa(x)$, M will output a correct w in expected time at most*

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

where access to P^* counts as one step only.

One can think of the error $\kappa()$ as the probability that one can convince the verifier without knowing a correct w. Being better than that requires some ability to actually compute w, and it gets more efficient, the better you are at convincing V. Thus this is based on the paradigm that a machine or an algorithm "knows" something if it can be used to compute it efficiently. Put differently: if you can pull the information out of P^*, it must somehow have been in there!

Based on what we said about Σ-protocols in the beginning, one should expect that any Σ -protocol for relation R is a proof of knowledge with knowledge error 2^{-t}, where t is the challenge length. Indeed this is true:

Theorem 1. *Let $\mathcal{P}$ be a Σ -protocol for relation R with challenge length t. Then $\mathcal{P}$ is a proof of knowledge with knowledge error 2^{-t}.*

Proof. Completeness is clear by definition.

For soundness, 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 $H_{\rho,e}$ is 1 if V accepts with this random choice and challenge, and 0 otherwise. Using P^* as black-box and choosing a random challenge, we can probe a random entry in H. By rewinding P^*, we can probe a random entry in the same row, i.e., where P^* uses the same internal random coins as before. Our goal is to find two 1's in the same row; using special soundness the resulting two conversations give us sufficient information to compute a witness w for x efficiently.

All we know is that $\epsilon := \epsilon(x)$ equals the fraction of 1-entries in H. Note that this gives no guarantees about the distribution of 1's in a given row. For instance, if we stumbled across a row with a single 1, we will never finish if we keep looking in that same row.

We can however make the following observation about this distribution. Define a row to be *heavy* if it contains a fraction of at least $\epsilon/2$ 1's. By a simple counting argument, we see that more than half of the 1's are located in heavy rows. Indeed, let H' be the sub-matrix of H consisting of all rows that are not heavy, and write h' for the total number of entries in H' and h for those in H. By assumption, the number of 1's in H is $h\epsilon$ and the number of 1's in H' is smaller than $h'\epsilon/2$. Then the number g of 1's in heavy rows satisfies

$$g > h\epsilon - h'\epsilon/2 \geq h\epsilon - h\epsilon/2 = h\epsilon/2.$$

Assume for the moment that

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

so that a heavy row contains at least two 1's. We repeatedly probe H at random, until we find a 1 entry, a "first hit." This happens after an expected number of $1/\epsilon$ tries.

By the observation above, with probability greater than 1/2, the first hit lies in a heavy row. *If* it does (note that we can't simply check this), and if we continue probing at random along its row, the expected number T of tries to find the second hit satisfies

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

tries. The inequality follows from the assumption on ϵ. On aggregate, we would be done in $O(1/\epsilon)$ tries, which is less than the time we are allowed to use (recall that we count access to P^* as one step).

However, with some probability smaller than $1/2$, the first hit is not in a heavy row. In that case we might spend too much time finding another 1 (if it exists!). To remedy this, we include an "emergency break."

For instance, we could restrict the number of attempts to find a second hit along a row to, say, $2T$. This way we never exceed the allotted time. Moreover, if the first hit is indeed in a heavy row, then the probability that we find the second in fewer than $2T$ tries is at least $1/2$. This follows from Markov's inequality. Thus, with probability $1/2 \cdot 1/2 = 1/4$ and using expected time $O(1/\epsilon)$ it finds two 1's along the same row. By repeating the whole procedure a constant number of times, we obtain an algorithm that achieves its goal in expected time $O(1/\epsilon)$, as desired.

The problem in this approach is that the algorithm needs to know the value of ϵ. This can be remedied by making the emergency break probabilistic. The picture is now as follows.

1. Probe random entries in H until the first 1 is found (the first hit).
2. Then start the following two processes in parallel, and stop when either one stops:

   Pr_1 Probe random entries in the row in which we found a 1 before, until another 1-entry is found (the second hit).

   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 done by probing a random entry in H and choosing a random number among $1, 2, ..., d$ - you output heads if the entry was a 1 and the number was 1.

Since d is constant, this algorithm certainly runs in expected time $O(1/\epsilon)$. We have to make sure though that Pr_1 gets enough time to finish before Pr_2 if indeed the first hit is in a heavy row.

Note that by the union bound, the probability that Pr_2 finishes after more than $d/(2\epsilon)$ tries is at least $1/2$. Now choose d "large", say d=16, so that with probability at least $1/2$, Pr_2 finishes after more than $8/\epsilon$ tries.

As before, if indeed the first hit is in a heavy row, then with probability greater than $1/2$, Pr_1 is done after fewer than $2T \leq 8/\epsilon$ tries.

Therefore, with probability greater than $1/2 \cdot 1/2 = 1/4$, Pr_1 finishes before Pr_2 in this case. Overall, this procedure finds two 1's along the same row with probability greater than $1/2 \cdot 1/4 = 1/8$. Repeating it a constant number of times, we obtain an algorithm that achieves its goal in expected time $O(1/\epsilon)$, as desired.

So what if $2^{-t} < \epsilon < 2^{-t+2}$?. We treat this case by a separate algorithm, using the fact that when ϵ is so small, we are in fact allowed time enough to probe an entire row. The algorithm we describe then simply runs in in parallel with the above algorithm.

Define δ by $\epsilon = (1+\delta)2^{-t}$; so that $0 < \delta < 3$. Let R be the number of rows in H. Then we have at least $(1+\delta)R$ 1's among the $R2^t$ entries. At most R of these can be alone in a row, thus at least δR of them must be in rows with at least two 1's. Such a row is called semi-heavy. The algorithm now does the following:

1. Probe random entries until a 1 is found.
2. Search the entire row for another 1 entry. If no such entry was found, go to step 1.

To analyze this, note that the fraction of ones in semi-heavy rows is $\delta/(1+\delta)$ among all ones and $\delta/2^t$ among all entries. The expected number of probes to find a 1 is $1/\epsilon = 2^t/(1+\delta)$. The expected number of probes to find a 1 in a semi-heavy row is $2^t/\delta$. So we expect to find a one in a semi-heavy row after finding $(1+\delta)/\delta$ 1's. For each 1 we find, we try the entire row, so we spend $O(2^t(1+\delta)/\delta)$ probes on this. In addition, we spend $O(2^t/\delta)$ probes on finding 1's in step 1, so altogether we spend

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

which is certainly $O(2^t/\delta)$. But this is no more than the time we are allowed:

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

4 The OR-proof

One basic construction with Σ-protocols allows a prover to show that given two inputs x_0, x_1, he knows w, such that either $(x_0, w) \in R$ or $(x_1, w) \in R$, BUT without revealing which is the case.

So we assume we are given a Σ-protocol $\mathcal{P}$ for R. Assume also that x_0, x_1 are common input to P, V, and that w is private input to P, where $(x_b, w) \in R$, where $b = 0$ or 1. Roughly speaking, the idea is that we will ask the prover to complete two instances of $\mathcal{P}$, with respect to x_0 resp. x_1. For x_b, he can do this for real, for x_{1-b} he will have to fake it using the simulator M. However, if we give him a little freedom in choosing the challenges to answer, he will be able to complete both instances. More precisely, consider the following protocol, which we call $\mathcal{P}_{OR}$:

1. P computes the first message a_b in $\mathcal{P}$, using x_b, w as input. P chooses e_{1-b} at random and runs the simulator M on input x, e_{1-b}, let $(a_{1-b}, e_{1-b}, z_{1-b})$ be the output. 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 $\mathcal{P}$ to challenge e_b using x_b, a_b, e_b, w as input. He sends e_0, z_0, e_1, z_1 to V.
4. V checks that $s = e_0 \oplus e_1$ and that conversations $(a_0, e_0, z_0), (a_1, e_1, z_1)$ are accepting conversations in $\mathcal{P}$, on inputs x_0 resp. x_1.

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

Theorem 2. *The protocol $\mathcal{P}_{OR}$ above is a Σ-protocol for R_{OR}. Moreover, for any verifier V^*, the probability distribution of conversations 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 has the right 3-move form. To verify soundness, let two accepting conversations

$$(a_0, a_1, s, e_0, e_1, z_0, z_1), (a_0, a_1, s', e_0', e_1', z_0', z_1') \text{ with } s \neq s'$$

be given. Then clearly it must be the case that for some $c = 0$ or 1, $e_c \neq e_c'$ and then from $(a_c, e_c, z_c), (a_c, e_c', z_c')$ we can compute w such that $(x_c, w) \in R$, by special soundness of $\mathcal{P}$.

Honest verifier zero-knowledge is clear: given s, choose e_0, e_1 at random subject to $s = e_0 \oplus e_1$ and run M twice, on inputs (x_0, e_0), resp. (x_1, e_1).

Finally assume we are given an arbitrary verifier V^*. Then observe that the distribution of conversations between P and V^* can be specified as follows: they have 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 $\mathcal{P}$ would choose them (this follows from perfect honest verifier zero-knowledge of $\mathcal{P}$). 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 has whatever distribution the honest prover in $\mathcal{P}$ outputs, given that the input was x_0 and the first part of the conversation was a_0, e_0. A similar conclusion holds for z_1. This is trivial for z_b and follows from perfect honest verifier zero-knowledge for z_{1-b}. It is clear from this specification that the distribution does not depend on b.

By the last claim in this theorem, the protocol $\mathcal{P}_{OR}$ is what is known as *witness indistinguishable* (WI): there are several different values of w that a prover may know that would enable him to complete the protocol successfully. But there is no way one can tell from the conversations which of the possible values he knows.

5 Additional Examples

As one obvious example of a Σ-protocol, one can note that the well-known protocol for graph isomorphism is in fact a Σ-protocol, albeit not a very efficient one: to have just one challenge bit, an entire graph, as large as the input graph, must be sent. What makes Schnorr's protocol so attractive is that for an input length of k bits, you can get challenge length k (and hence error probability 2^{-k}) at a communication cost linear in k. Here are a couple of examples with similar efficiency:

Let p, q be chosen as in Schnorr's protocol, and let $g, \bar{g}, h, \bar{h} \in Z_p^*$ be of order q. Assume P gets as input w where $h = g^w \bmod p$, $\bar{h} = \bar{g}^w \bmod p$. Consider the following protocol:

1. P chooses r at random in Z_q and sends $a = g^r \bmod p, \bar{a} = \bar{g}^r \bmod p$ to V.
2. V chooses a challenge e at random in Z_{2^t} and sends it to P. Here, t is fixed such that $2^t < q$.
3. P sends $z = r + ew \bmod q$ to V, who checks that $g^z = ah^e \bmod p$ and $\bar{g}^z = \bar{a}\bar{h}^e \bmod p$, that p, q are prime that $g, \bar{g}, h, \bar{h}$ have order q and accepts iff this is the case.

Exercise Prove that this is a Σ-protocol for equality of discrete logs, more precisely show that this is a Σ-protocol for the relation

$$\{(x, w)|\ x = (p, q, g, \bar{g}, h, \bar{h}) \textit{ and } h = g^w, \bar{h} = \bar{g}^w\}.$$

- here it is understood that it should also be satisfied that p, q are prime, that $w \in Z_q$, and that $g, h, \bar{g}, \bar{h} \in Z_p^*$ have order q.

Let n be an RSA modulus and q be a prime. Assume we are given some element $y \in Z_n^*$, and P knows an element w such that $w^q = y \bmod n$. Consider the following protocol:

1. P chooses r at random in Z_n^* and sends $a = r^q \bmod n$ to V.
2. V chooses a challenge e at random in Z_{2^t} and sends it to P. Here, t is fixed such that $2^t < q$.
3. P sends $z = rw^e \bmod n$ to V, who checks that $z^q = ay^e \bmod p$, that q is a prime, that $gcd(a, n) = gcd(y, n) = 1$, and accepts iff this is the case.

Exercise Prove that this is a Σ-protocol for proving knowledge of q'th roots modulo n, more precisely show that this is a Σ-protocol for the relation

$$\{(x, w)|\ x = (n, q, y), y, w \in Z_n^*, q \textit{ prime},\ \textit{and } y = w^q \bmod n\}.$$

Hint: if you are given e, e' such that $e - e' \neq 0 \bmod q$, then since q is a prime, this means that $gcd(q, e-e') = 1$, and therefore there exist integers α, β such that $\alpha q + \beta(e - e') = 1$.

6 Hard Relations

Of course, if it was easy for a given relation R and input x to find w with $(x, w) \in R$, there would not be much point in having the verifier talk to the prover. So we define a relation to be hard, if one can efficiently generate (x, w)'s such that, when given only x, finding a witness w for it is hard. More precisely:

Definition 3. *A relation R is said to be hard, if*

- *There exists a probabilistic polynomial time algorithm G, called the* generator*, which on input 1^k outputs a pair $(x, w) \in R$ where $|x| = k$.*
- *The following holds for all probabilistic polynomial time algorithms A: consider the experiment where we run G on input 1^k, give the x produced to A, and let w_A be the output A produces. Let $p_A(k)$ be the probability that $(x, w_A) \in R$. Then $p_A(k)$ is negligible in k.*

So for instance if we define a generator that picks random primes p and q such that $q|p - 1$ and random $g, h \in Z_p^*$ with $ord(g) = ord(h) = q$, then saying that the example relation DL is hard with respect to this generator is equivalent to the standard discrete log assumption.

With this concept, we can show more about the protocol $\mathcal{P}_{OR}$ we constructed above. Although it cannot be shown to be zero-knowledge when the challenge length is large (no Σ-protocol can), it has some security against a cheating verifier when based on a hard relation: it turns out that even a cheating verifier cannot, by talking to the prover, learn enough to be able to compute a solution to any of the two public problem instances.

To state this more precisely, let a hard relation R with generator G be given and assume R has Σ-protocol $\mathcal{P}$. Consider the protocol $\mathcal{P}_{OR}$ for R_{OR} that we constructed above. Consider the following game against an arbitrary poly-time verifier V^*:

– Run G twice on input 1^k to get pairs $(x_0, w_0), (x_1, w_1)$. Choose bit b at random and give w_b as private input to the prover P in $\mathcal{P}_{OR}$.
– Let V^* execute $\mathcal{P}_{OR}$ with P an arbitrary (polynomial) number of times on common input x_0, x_1.
– V^* outputs a string w^*.

We are interested in the probability that $(x_0, w^*) \in R$ or $(x_1, w^*) \in R$, and we say that V^* wins the game if this happens. We have

Theorem 3. *The probability that V^* wins the above game is negligible in k.*

Proof. Consider the following algorithm which gets x of length k as input (as generated by G) and uses V^* to try to find a w such that $(x, w) \in R$. It does the following:

1. Run G on input 1^k to get $(\tilde{x}, \tilde{w}) \in R$.
2. Choose b at random, and set $(x_0, x_1) = (x, \tilde{x})$ if $b = 0$ and $(x_0, x_1) = (\tilde{x}, x)$ otherwise.
3. Now we let V^* conduct its interaction with the prover in $\mathcal{P}_{OR}$. Here we play the role of the prover, following exactly the prover's algorithm and using our knowledge of $\tilde{w}$. Once V^* is done, we output the w^* that V^* produces.

Let ϵ be the probability with which V^* wins. Now, witness indistinguishability of $\mathcal{P}_{OR}$ implies that V^* has no information on our choice of b. So since we choose b at random, it is clear that the probability that $(x, w^*) \in R$, given that V^* wins, is at least $1/2$. And hence the probability that we reach our goal is at least $\epsilon/2$. But since R is hard, this probability is negligible, and so ϵ is negligible too.

This shows that $\mathcal{P}_{OR}$ has a property known as *Witness Hiding* (WH): Even by cheating, the verifier cannot learn enough information to be able to compute the prover's secret. This is a weaker property than zero-knowledge which demands that the verifier learns nothing new whatsoever. But still WH is good enough in many cases, such as for instance in the obvious application for identification schemes which we look at in the next section.

It is interesting to note that even though $\mathcal{P}_{OR}$ is WH, this does not have to be the case for the protocol $\mathcal{P}$ we start from. Thus the OR construction is a very economic way of producing extra security: the complexity is only about twice that of $\mathcal{P}$.

7 Identification Schemes from Σ-protocols

We want now construct a secure identification scheme for users $U_1, .., U_n$. Given a hard relation R with Σ-protocol $\mathcal{P}$, this is easily done: to set up the system, we run the generator G n times on input 1^k to get pairs $(x_1, w_1), ..., (x_n, w_n)$. We give w_i as private key to user U_i and publish the list of x_i's.

In order to identify himself, U_i executes $\mathcal{P}$ with x_i as public input, playing the role of the prover. Here, we assume that $\mathcal{P}$ has been constructed such that the length of a challenge is k bits (or at least $\theta(k)$), so that we are sure that there is an exponentially large number of challenges. This is always possible, as we have seen. We also assume that $\mathcal{P}$ is WH. As we have seen this can be achieved by the OR-construction, if $\mathcal{P}$ does not have this property already.

Now, by completeness, U_i will always succeed. Moreover, the soundness and WH properties imply that any adversary A who first gets to do the protocol as verifier with U_i (following any strategy he wants) still cannot do the protocol as prover with non-negligible success probability. Or more precisely, if he could, this would imply that he could in fact compute w_i on his own. To see this, consider the following algorithm:

1. Let A play verifier against U_i as many times as he wants.
2. Consider the knowledge extractor M guaranteed to exist by the fact that the protocol is a proof of knowledge with soundness error 2^{-k} (see Theorem 1). Let M interact with A (in its current state after having talked to the honest prover)
3. Let w_i be M's output. Output w_i.

If A's success probability when acting as prover is non-negligible, say greater than $1/f(k)$ for some polynomial $f()$, then this algorithm returns a correct w_i in expected polynomial time. This is because the definition of knowledge soundness guarantees that M succeeds in expected time proportional to $\frac{1}{1/f(k)-2^{-k}}$ wich is certainly less than a polynomial for all large enough k (and is in fact approximately equal to $f(k)$).

In other words, this is an efficient algorithm that first interacts with the prover on input x_i and then computes w_i such that $(x_i, w_i) \in R$, and this contradicts the WH property of $\mathcal{P}$.

There is one potential problem in practice with this: we have assumed that the attack takes place in two phases: first A tries to get information from the honest prover U_i, and then A tries to impersonate the prover towards the honest verifier. In this last phase A is on his own and cannot get help from the honest prover. This may not always be the case in real life. Suppose U_i is identifying himself to a corrupt verifier $\tilde{V}$, who is colluding with A *while* he is trying to impersonate U_i towards honest verifier V. Now, every time U_i sends a message to $\tilde{V}$, this is immediately relayed to A who sends it to V, while messages from V are relayed in a similar way back to U_i. Clearly, this will lead V to accept. Now, if for instance U_i thinks he is making a purchase in some store (say, owned by $\tilde{V}$), the attack could have the effect that A is in fact spending U_i's money somewhere else. This is known as the mafia-fraud or the man-in-the-middle attack.

So is this identification scheme flawed? Actually, it isn't! Notice that the protocol correctly demonstrates to V that U_i is alive "somewhere out there". By just relaying messages from one party to the other, the two cheating parties are essentially reducing themselves to a communcation channel between the honest prover and verifier. The protocol cannot be held responsible for the fact that the communcation channel turns out to be different from what the honest players might have expected.

Put differently, the problem is not that the protocol is wrong, but rather that the security it was designed to achieve is not sufficient for the practical scenario sketched above. Some thought on the scenario will show that the problem comes from the fact that the prover has no way to distinguish one verifier from the other: we have not assumed that verifiers have identities or public keys. We therefore have to modify the set-up and protocol, in order to change this.

To this end, we will assume that provers *as well as* verifiers have public keys, and (as before) that it is always possible to determine the public key of a player with a given name. We then modify the protocol as follows: When prover U_i wants to identify himself to U_j, they first exchange their identities and determine each other's public keys x_i, x_j. Now, instead of proving that he knows w_i, U_i will use the OR-construction to prove that he knows w_i or w_j. Note that this proof will be convincing to an honest U_j: he knows w_j himself and no one else does, so the only other prover who could do this protocol successfully is U_i.

Now consider the scneario from before: A is trying to impersonate U_i towards honest U_v, and simultaneously U_i is proving his identity to corrupt $\tilde{U_j}$. Note first that U_v is expecting to see a proof that A knows w_i or w_v. The proof given simultaneously by U_i does not directly help because it is different, nemaly a proof of knowledge of w_i or w_j. In fact, it's even better: even though $\tilde{U_j}$ is corrupt, we can reasonably assume that he knows his own private key w_j. With this knowledge, the proof given by U_i can be simulated perfectly, by WI of the protocol: knowledge of w_i or w_j will suffice to do it, and one cannot tell which one is used. So in fact, the attack can be carried out without U_i participating at all, and we have already seen that no attack of this type can be successful.

One way to describe this idea is that it gives a way for the prover to target his proof at a particular verifier such that only this party will be convinced. It is important to understand that the solution only works in practice if the prover can correctly determine *who* to target. If for instance the protocol is carried out by the prover's mobile phone in some shop, it is not clear that the phone could on its own determine which shop it is in. It will most likely receive a name and certified public key from the shop's machine. It should then on the display show the name to the human user who can interrupt if the name does not correspond to the physical shop we are in.

8 Zero-Knowledge from Σ-protocols

For completeness, we sketch a construction allowing to make an efficient zero-knowledge protocol from Σ protocol $\mathcal{P}$ for a hard relation R. As above, we may assume without loss of generality that $\mathcal{P}$ is WH. Now suppose we have public input x and private input w to prover P. The protocol goes as follows:

1. V runs generator G on input 1^k, where k is the length of x, to get $(x', w') \in R$.
2. V sends x' to P and proves using $\mathcal{P}$ that he knows w'. Note that here V plays the role as prover, and P plays verifier.
3. If P accepted in the previous step, P uses the OR-construction to prove that he knows w or w'.

We only sketch the proof that this works. Zero-knowledge: consider a simulator playing against a verifier V^*. Roughly speaking, if V^*'s strategy is such that he convinces P in step 2 with only negligible probability, then protocol and simulation stops there essentially always. Otherwise, we can use rewinding of V^* to extract a w' that is good w.r.t. x' in the same way as in the previous section. Then step 3 can be simulated by just following the protocol using the knowledge of w'. By WI, no one can tell the difference to what the prover does, despite the fact that he uses w to do the proof.

Soundness is still OK, if we adopt a "cryptographically secure" variant of soundness: to extract w from some successful prover P^*, we just follow V's algorithm in steps 1-2, and then use rewinding in step 3 to extract a value w^*, which must satisfy that $(x, w^*) \in R$ or $(x', w^*) \in R$. However, the latter case only happens with negligible probability because the proof given in step 2 is WH and R is a hard relation. So this shows that P^* must know w to be successful. Note that by interleaving the moves, a total of 4 moves suffices.

We note without proof that there is an equally efficient variation of this protocol that satisfies soundness as we have defined it earlier in these notes.

9 Commitment Schemes from Σ-protocols

Assume we are given a hard relation R with generator G and Σ protocol $\mathcal{P}$. Assume also that it is easy to check membership in L_R, that is, given x, it is easy to decide if there exists w such that $(x, w) \in R$.

With this set-up, we can build a perfectly hiding commitment scheme, which is efficient and allows commitment to many bits, if $\mathcal{P}$ is efficient:

Set-up V runs (in private) generator G on input 1^k to get $(x, w) \in R$, sends x to P who checks that $x \in L_R$.

Commit To commit to a t-bit string e, P runs the simulator M on input x, e to get (a, e, z), and sends a to V.

open To open the commitment, P sends e, z to V, who checks that (a, e, z) is an accepting conversation (w.r.t. x).

Theorem 4. *The above scheme is a perfectly hiding commitment scheme with computational binding.*

Proof. For the hiding part, note that in real life, P's first message is independent of the challenge e. Since $x \in L_R$, simulation by M is perfect by definition of Σ-protocols, so hence the a generated by M is uncorrelated to e. For the binding, if some P^* could efficiently output a, and open it both as e, z and as e', z' with $e \neq e'$, then we would have accepting conversations in $\mathcal{P}$, $(a, e, z), (a, e', z')$, this means by definition that we can compute w efficiently, and this contradicts the assumption that R is a hard relation.

10 Making Σ-protocols non-interactive using random oracles

Imagine a world where all protocol participants have access to a *random oracle*. This is an entity that initially chooses (in private) a random function $R : \{0,1\}^l \rightarrow \{0,1\}^t$ for some l, t. Then any player can send any bit string a of length l to the oracle which will then return $R(a)$. Since R was completely random, $R(a)$ is a uniformly chosen string of length t, and is independent of a. Moreover, seeing the value of $R(a)$ gives no advantage whatsoever in predicting the value $R(b)$ for $b \neq a$. However, R is fixed, so every time someone sends a to the oracle, the answer will be the same value $R(a)$. This is known as the *random oracle model*.

It is clear that with such an oracle, the prover in a Σ-protocol can execute the protocol without talking to a verifier, one can just replace the verifier's random choice of challenge by sending the first message to the oracle, and using the response as challenge. If this generated the "conversation" (a, e, z), the prover can send (a, z) to the verifier in one message. The verifier calls the oracle with a as input to get the value of e, and checks the answer z as it would have done normally.

Could a cheating prover convince the verifier in this game? First thing to realize is that the prover cannot get an oracle response on a without calling the oracle, and so in fact he has no information about e before he has sent a. So this is completely equivalent to talking to a real verifier. The only difference is that a cheating prover is free to call the oracle as many times as he wants in the hope of getting some challenge he can answer. But if the number of challenges is exponentially large, and the prover only has polynomial time, this is not a feasible strategy.

Also, it is clear that using the random oracle removes any influence from a cheating verifier, in fact it is equivalent to forcing the verifier to be honest, because the challenge values are always randomly and independently chosen, just like the honest verifier would choose them. And since we have assumed that Σ-protocols are honest verifier zero-knowledge, Σ-protocols are automatically zero-knowledge in the random oracle model.

More formally speaking, the definition of zero-knowledge in this model says that the simulator is allowed to decide what the oracle responses should be, as long as they have the same distribution as in real life. So then one can just choose e at random and run the normal simulator M to get (a, e, z), and define the oracle's response on input a to be e, and output (a, z).

This type of construction can also be used to make secure signature schemes from Σ-protocols: to generate keys we make a pair (x, w) in a hard relation, and let x be the public key, and w the private key. To sign a message m, the signer runs the Σ-protocol (in the role of the prover) computing the first message a. He then calls the random oracle with a, m as input, and takes the answer e as the challenge. Using his knowledge of w, he can compute the answer z. The signature is then the pair a, z. In the random oracle model, one can then prove that breaking this signature scheme is as hard as computing w from x.

It is reasonable to ask what (if anything) all this has to do with real life, since obviously no realistic implementation could hope to rely on such a random oracle? There is no final answer to this, but from a heuristic point of view it seems that many of the standard one-way hash functions in use today, such as $SHA1$ or $RIPEMD160$ could be used in place of the random oracle: because of their one-wayness and complexity it seems that an adversary would have to decide on an input and compute the function before being able to do any sensible computation that depends on the output. This is why random oracles are sometimes referred to as idealized hash functions. But it is important to realize that assuming that a hash function is "equivalent" to a random oracle is a heuristic assumption that cannot be formally proved: the hash function is not random, it is fixed, known and is not hidden inside an oracle.

In other words, there is no hope of being able to prove that some concrete hashfunction could be used to make all Σ protocols non-interactive in a secure way. On the other hand it may well be possible to show that using a concrete hash function together with a concrete Σ-protocol actually works. But no result of this type is known.

Despite these theoretical problems, hash functions are being used in this way in practice to large extent, simply because the practical advantages of being able to remove interaction, and because the random oracle model also allows construction of secure signature and encryption schemes with better efficiency than what is otherwise known. On the positive side, it should also be noted that most, if not all attacks on practical protocols using hash functions as subroutines in fact treat the hash function as if it were a random oracle: one thinks of it as a black box outputting random values. The point now is that a security proof in the random oracle model does in fact rule out any attack of this type.

11 Notes

The first efficient Σ-protocol was proposed by Schnorr [10] (the first discrete log example in this note). The RSA based example from the exercises was proposed by Guillou and Quisquater [9]. Damgård [3] showed how to make commitment schemes from Σ protocols. None of these papers actually use the concept as such of Σ protocols. This idea of Σ-protocols as an abstract concept was introduced by Cramer in his PhD thesis [4]. The OR-proof is due to [5] and the identification that is secure against man-in-the-middle attacks in a public key scenario is due to [7]. In [6] it is shown how to to use Σ-protocols for signatures without relying on random oracles. The idea of using a hash function in place of the verifier to do non-interactive proofs comes from Fiat and Shamir [8], this was later formalized as the random oracle model by Bellare and Rogaway [2].

References

1. M. Bellare and O. Goldreich: *On defining proofs of knowledge*: proc. of Crypto 92.
2. M. Bellare and P. Rogaway: *Random Oracles are Practical*, Proc. of ACM CCS conference, 1993.
3. I. Damgård: *On the Existence of Bit Commitment Schemes and Zero-Knowledge Proofs*, Proc. of Crypto 89
4. R.Cramer: *Modular Design of Secure, yet Practical Cryptographic Protocols*, PhD Thesis, University of Amsterdam, 1996
5. R. Cramer, I. Damgård, B. Schoenmakers: *Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols*, Proc. of Crypto '94.
6. R. Cramer, I. Damgård: *Secure Signatures from Interactive Protocols*, Proc. of Crypto '95.
7. R. Cramer, I. Damgård: *Fast and Secure Immunization against Man-in-the-Middle Impersonations*, Proc. of EuroCrypt '97.
8. A. Fiat and A. Shamir: *How to Prove Yourself: Practical Solutions to the Identification and Signature Problem*, Proc. of Crypto 86.
9. L. Guillou and J.-J. Quisquater: *A Practical Zero-Knowledge Protocol fitted to security microprocessor minimizing both transmission and memory*, Proc. of EuroCrypt 88.
10. C. Schnorr: *Efficient Signature Generation by Smart Cards*; Journal of Cryptology vol. 4 (3) 1991.