2.4 The Schnorr Identification Scheme

Another approach to identification schemes is to design schemes "from scratch", without using any other cryptographic tools as building blocks. This would be done primarily to obtain more efficient protocols. Such schemes typically involve having someone identify themselves by proving that they know the value of some secret quantity (i.e., a private key) without having to reveal its value. The *Schnorr Identification Scheme* is an example of such a scheme; it is based on the Discrete Logarithm problem, which is believed to be difficult to solve in certain groups. We introduce this problem now.

Let $(G, \cdot)$ be a finite abelian group. For an element $\alpha \in G$ having order n, define

$$\langle\alpha\rangle = \{\alpha^i : 0 \leq i \leq n-1\}.$$

$\langle\alpha\rangle$ is a cyclic subgroup of G having order n, and α is a generator of $\langle\alpha\rangle$. Here is the formal definition of the Discrete Logarithm problem.

Problem 2.1:	Discrete Logarithm
Instance:	A finite multiplicative group $(G, \cdot)$, an element $\alpha \in G$ having order n, and an element $\beta \in \langle\alpha\rangle$.
Question:	Find the unique integer a, $0 \leq a \leq n-1$, such that $\alpha^a = \beta$. We will denote this integer a by $\log_\alpha \beta$.

A frequently used setting for the Discrete Logarithm problem is to take α to be an element having prime order q in the group $\mathbb{Z}_p{}^*$ (where p is prime and $p - 1 \equiv 0 \pmod{q}$). Such an element α can be obtained by raising a primitive element in $\mathbb{Z}_p{}^*$ to the $(p-1)/q$th power. This setting of the Discrete Logarithm problem is thought to be secure if $p \approx 2^{1024}$ and $q \approx 2^{160}$.

REMARK An exponentiation operation in a large group, say $y = f(a) = \alpha^a$, can be performed efficiently using the SQUARE-AND-MULTIPLY algorithm. The discrete logarithm $a = \log_\alpha y$ is just the inverse function evaluated at y, i.e., $f^{-1}(y)$. The fact that computing the function f is easy, while computing the inverse function f^{-1} is apparently difficult (in certain groups) is the basis for many cryptographic protocols. ▮

We now describe the *Schnorr Identification Scheme*, which is one of the most attractive practical identification schemes. The scheme requires a trusted authority, or TA, who chooses some common system paramters (*domain parameters*) for the scheme, as follows:

1. p is a large prime (i.e., $p \approx 2^{1024}$).

2. q is a large prime divisor of $p-1$ (i.e., $q \approx 2^{160}$).
3. $\alpha \in \mathbb{Z}_p{}^*$ has order q.
4. t is a *security parameter* such that $q > 2^t$. (For most practical applications, $t = 40$ will provide adequate security.)

The domain parameters p, q, α and t are all made public, and they will be used by everyone in the network.

Every user in the network chooses their own private key, a, where $0 \leq a \leq q-1$, and constructs a corresponding public key $v = \alpha^{-a} \bmod p$. The TA issues certificates for everyone in the network. Each user's certificate will contain their public key (and, perhaps, the public domain parameters). This information, as well as the user's identifying information, is signed by the TA, of course.

The *Schnorr Identification Scheme* is a three-flow protocol in which Alice identifies herself to Bob, say, by executing Protocol 2.8.

Protocol 2.8: SCHNORR IDENTIFICATION SCHEME

1. Alice chooses a secret random number k, where $0 \leq k \leq q-1$, and computes
$$\gamma = \alpha^k \bmod p.$$
Alice sends her certificate, $\mathbf{Cert}(Alice)$, and γ to Bob. (The value γ is called a *commitment*.)
2. Bob verifies the signature of the TA on Alice's certificate, $\mathbf{Cert}(Alice)$, thereby authenticating Alice's public key. Then Bob chooses a random *challenge*, r, where $1 \leq r \leq 2^t$, and gives r to Alice.
3. Alice computes
$$y = k + ar \bmod q$$
and gives y to Bob.
4. Bob verifies that
$$\gamma \equiv \alpha^y v^r \pmod{p}.$$
If so, then Bob outputs "accept"; otherwise, Bob outputs "reject".

The following congruences demonstrate that Alice will be able to prove her identity to Bob, assuming that both parties are honest and perform correct computations:

$$\begin{aligned}
\alpha^y v^r &\equiv \alpha^{k+ar} v^r \pmod{p} \\
&\equiv \alpha^{k+ar} \alpha^{-ar} \pmod{p} \\
&\equiv \alpha^k \pmod{p} \\
&\equiv \gamma \pmod{p}.
\end{aligned}$$

The fact that Bob will accept Alice's proof of identity (assuming that he and Alice are honest) is sometimes called the *completeness* property of the protocol.

Let's work out a small, toy example. The following example omits the authentication of Alice's public key by Bob.

Example 2.1 Suppose $p = 88667$, $q = 1031$ and $t = 10$. The element $\alpha = 70322$ has order q in $\mathbb{Z}_p{}^*$. Suppose Alice's private key is $a = 755$; then

$$\begin{aligned} v &= \alpha^{-a} \bmod p \\ &= 70322^{1031-755} \bmod 88667 \\ &= 13136. \end{aligned}$$

Now suppose Alice chooses the random number $k = 543$. Then she computes

$$\begin{aligned} \gamma &= \alpha^{k} \bmod p \\ &= 70322^{543} \bmod 88667 \\ &= 84109. \end{aligned}$$

and sends γ to Bob. Suppose Bob issues the challenge $r = 1000$. Then Alice computes

$$\begin{aligned} y &= k + ar \bmod q \\ &= 543 + 755 \times 1000 \bmod 1031 \\ &= 851 \end{aligned}$$

and sends y to Bob. Bob then verifies that

$$84109 \equiv 70322^{851} 13136^{1000} \pmod{88667}.$$

Finally, Bob outputs "accept". $\square$

The *Schnorr Identification Scheme* was designed to be very fast and efficient, both from a computational point of view and in the amount of information that needs to be exchanged in the protocol. It is also designed to minimize the amount of computation performed by Alice, in particular. This is desirable because, in many practical applications, Alice's computations will be performed by a smart card with low computing power, while Bob's computations will be performed by a more powerful computer.

Let us consider Alice's computations. Step 1 requires an exponentiation (modulo p) to be performed; step 3 comprises one addition and one multiplication (modulo q). It is the modular exponentiation that is computationally intensive, but this can be precomputed offline, before the protocol is executed, if desired. The online computations to be performed by Alice are very modest.

It is also a simple matter to calculate the number of bits that are communicated during the protocol. We depict the information that is communicated (excluding Alice's certificate) in the form of a diagram:

$$
\begin{array}{ccc}
\text{Alice} & & \text{Bob} \\[1em]
\begin{array}{c} k \in_R \{0, \ldots, q-1\} \\ \gamma = \alpha^k \bmod p \end{array} & \xrightarrow{\quad \gamma \quad (1024 \text{ bits}) \quad} & \\[1em]
 & \xleftarrow{\quad r \quad (40 \text{ bits}) \quad} & r \in_R \{1, \ldots, 2^t\} \\[1em]
y = k + ar \bmod q & \xrightarrow{\quad y \quad (160 \text{ bits}) \quad} & \gamma \equiv \alpha^y v^r \pmod{p}?
\end{array}
$$

In the above diagram, the notation $\in_R$ is used to denote a random choice made from a specified set.

Alice gives Bob 1024 bits of information (excluding her certificate) in the first flow; Bob gives Alice 40 bits in the second flow; and Alice gives Bob 160 bits in the third flow. So the communication requirements are quite modest, as well.

The information transmitted in the second and third flows of the protocol have been reduced by the way in which the scheme is designed. In the second flow, a challenge could be taken to be any integer between 0 and $q-1$; however, this would yield a 160-bit challenge. We will argue a little bit later that a 40-bit challenge provides sufficient security for most applications.

In the third flow, the value y is an exponent. This value is only 160 bits in length because the scheme is working inside a subgroup of $\mathbb{Z}_p{}^*$ of order $q \approx 2^{160}$. This permits the information transmitted in the third flow to be reduced significantly, as compared to an implementation of the scheme in the "whole group", $\mathbb{Z}_p{}^*$, in which an exponent would be 1024 bits in length. According the current state of knowledge regarding algorithms for the Discrete Logarithm problem, the security of the scheme is not reduced by restricting the scheme to the subgroup of the specified order, q.

The first flow clearly requires the most information to be transmitted. One possible way to reduce the amount of information is to replace the 1024-bit value γ by a 160-bit message digest, $\gamma' = \mathit{SHA\text{-}1}(\gamma)$. Then, in the last step of the protocol, Bob would verify that (the message digest) $\gamma = \mathit{SHA\text{-}1}(\alpha^y v^r \pmod{p})$.

Let's now study the security of the scheme. As mentioned previously, t is a security parameter. It is sufficiently large to prevent an impostor posing as Alice, say Olga, from guessing Bob's challenge, r. (If Olga guessed the correct value of r, she could choose any value for y and precompute

$$\gamma = \alpha^y v^r \bmod p.$$

She would give Bob the value γ in the first flow of the protocol, and when she receives the challenge r, she would supply the value y she has already chosen. Then the congruence involving γ would be verified by Bob, and he would "accept".) The probability that Olga will guess the value of r correctly is 2^{-t} if r is chosen at random by Bob. Thus, $t = 40$ should be a reasonable value for most applications.

Notice that Bob should choose a new, random challenge, r, every time Alice identifies herself to him. If Bob always used the same challenge r, then Olga could impersonate Alice by the method we just described.

Alice's computations in the scheme involve the use of her private key, a. The value a functions somewhat like a PIN, in that it convinces Bob that the person (or entity) carrying out the identification protocol is, indeed, Alice. But there is an important difference from a PIN: in this identification protocol, the value of a is not revealed. Instead, Alice (or more accurately, Alice's smart card) "proves" that she/it knows the value of a in the third flow of the protocol, by computing the correct response, y, to the challenge, r, issued by Bob. An adversary could attempt to compute a, because a is just a discrete logarithm: $a = -\log_\alpha v$ in $\mathbb{Z}_p{}^*$. However, we are assuming that this computation is infeasible.

We have argued that Olga can guess Bob's challenge, r, and thereby impersonate Alice, with probability 2^{-t}. Suppose that Olga can do better than this. If Olga can impersonate Alice successfully with a probability exceeding 2^{-t}, then it means that Olga knows a value γ (a commitment of her choice), and two possible challenges, r_1 and r_2, such that she can compute responses y_1 and y_2, respectively, which would cause Bob to accept. Therefore, Olga knows values r_1, r_2, y_1 and y_2 such that

$$\gamma \equiv \alpha^{y_1} v^{r_1} \equiv \alpha^{y_2} v^{r_2} \pmod{p}.$$

It follows that

$$\alpha^{y_1 - y_2} \equiv v^{r_2 - r_1} \pmod{p}.$$

It holds that $v \equiv \alpha^{-a} \pmod{p}$, where a is Alice's private key. Hence,

$$\alpha^{y_1 - y_2} \equiv \alpha^{-a(r_2 - r_1)} \pmod{p}.$$

The element α has order q, so it must be the case that

$$y_1 - y_2 \equiv a(r_1 - r_2) \pmod{q}.$$

Now, $0 < |r_2 - r_1| < 2^t$ and $q > 2^t$ is prime. Hence $\gcd(r_2 - r_1, q) = 1$, and therefore $(r_1 - r_2)^{-1} \bmod q$ exists. Hence, Olga can compute Alice's private key, a, as follows:

$$a = (y_1 - y_2)(r_1 - r_2)^{-1} \bmod q.$$

The above analysis proves that anyone who is able to successfully impersonate Alice with a probability exceeding 2^{-t} must know (or be able to easily compute) Alice's private key, a. It therefore follows, roughly speaking, that being able to impersonate Alice is equivalent to knowing Alice's private key. (We proved above that a "successful" impersonator can compute a. Conversely, it is obvious that anyone who knows the value of a can impersonate Alice, with probability equal to 1.) A protocol satisfying this type of security property is called a *proof of knowledge*.

We provide an example to illustrate the above discussion.

Example 2.2 Suppose we have the same parameters as in Example 2.1: $p = 88667, q = 1031, t = 10, \alpha = 70322$ and $v = 13136$. Suppose, for the commitment $\gamma = 84109$, that Olga is able somehow to determine two correct responses: $y_1 = 851$ is the correct repsonse for the challenge $r_1 = 1000$; and $y_1 = 454$ is the correct repsonse for the challenge $r_1 = 19$. In other words,

$$84109 \equiv \alpha^{851} v^{1000} \equiv \alpha^{454} v^{19} \pmod{p}.$$

Then Olga can compute

$$a = (851 - 454)(1000 - 19)^{-1} \bmod 1031 = 755,$$

and thus discover Alice's private key. ▯

We have proved that the protocol is a proof of knowledge. But this is not sufficient to ensure that the protocol is "secure". We still need to consider possibility that secret information (namely, Alice's private key) might be released to a verifier who takes part in the protocol, or an observer. Our hope is that no information about a will be gained by Olga when Alice proves her identity. If this is true, then Olga will not be able subsequently to masquerade as Alice (assuming that the computation of the discrete logarithm a is infeasible).

In general, we could envision a situation whereby Alice proves her identity to Olga, say, on several different occasions. After several sessions of the protocol, Olga will try to determine the value of a so she can subsequently impersonate Alice. If Olga can determine no information about the value of a by taking part in a "reasonable" number of sessions of the protocol as the verifier, and then performing a "reasonable" amount of computation, then the protocol is termed a *zero-knowledge* protocol. This would prove that the protocol is secure, under the assumption that a is infeasible to compute. (Of course, it would necessary to define in a precise way the term "reasonable", in order to have a meaningful statement of security.)

We will show that the Schnorr Identification Scheme is *zero-knowledge for an honest verifier*, where an honest verifier is defined to be one who chooses his or her challenges r at random, as specified by the protocol.

We require the notion of a *transcript* of a session, which consists of a triple $T = (\gamma, r, y)$ in which $\gamma \equiv \alpha^y v^r \pmod{p}$. The verifier (or an observer) can obtain a transcript $T(S)$ of each session S. The set of possible transcripts is

$$\mathcal{T} = \{(\gamma, r, y) : 1 \leq r \leq 2^t, 0 \leq y \leq q-1, \gamma \equiv \alpha^y v^r \pmod p\}.$$

It is easy to see that $|\mathcal{T}| = q\,2^t$. Further, it is not difficult to prove that the probability that any particular transcript occurs in any given session is $1/(q\,2^t)$, assuming that the challenges r are generated at random. We argue this as follows: for any fixed value of r, there is a one-to-one correspondence between the value of $\gamma \in \langle\alpha\rangle$ and the value of $y \in \{0, \ldots, q-1\}$ on a particular transcript. We are assuming that Alice chooses γ at random (namely, by choosing a random k and computing $\gamma = \alpha^k \bmod p$), and we also assume that Bob chooses r at random. These two values determine the value of y. Since there are q possible choices for γ and 2^t possible choices for r, it follows that all possible transcripts occur with the same probability, $1/(q\,2^t)$, in sessions involving an honest verifier.

The key point of the zero-knowledge aspect of the protocol is that Oscar (or anyone else) can generate simulated transcripts, having exactly the same probability distribution as real transcripts, without taking part in the protocol. This is done by the following three simple steps:

1. choose $r \in_R \{1, \ldots, 2^t\}$
2. choose $y \in_R \{0, \ldots, q-1\}$
3. compute $\gamma = \alpha^y v^r \bmod p$

It is easy to see that the probabilty that any $T \in \mathcal{T}$ is generated by the above procedure is $1/(q\,2^t)$.

What is the significance of the fact that transcripts can be simulated? We claim that, whatever an honest verifier can compute after taking part in several sessions of the protocol, he can alternatively compute the same information without taking part in any sessions of the protocol. In particular, computing Alice's private key, a, which is necessary for Oscar to be able to impersonate Alice, is not made easier for Oscar if he plays the role of the verifier in one or more sessions in which he chooses his challenges randomly.

The above statements can be justified further, as follows. Suppose there exists an algorithm EXTRACT which, when given a set of transcripts, say $T_1, \ldots, T_\ell$, computes a private key, say a, with some probability, say ϵ. We assume that the transcripts are actual transcripts of sessions, in which the participants follow the protocol. Suppose that $T_1', \ldots, T_\ell'$ are simulated transcripts. We have noted that the probability distribution on simulated transcripts is identical to the probability distribution on real transcripts. Therefore EXTRACT$(T_1', \ldots, T_\ell')$ will also compute a with probability ϵ. This establishes that executing the protocol does not make computing a easier, so the protocol is zero-knowledge.

Let's consider the possibility that Oscar (a "dishonest verifier") might obtain some useful information by choosing his challenges r in a non-uniform way. To be specific, suppose that Oscar chooses his challenge r using some function that depends, in a complicated way, on Alice's commitment, γ. There does not seem to be any way to perfectly simulate the resulting probability distribution on transcripts, and therefore we cannot prove that the protocol is zero-knowledge in the way that we did for an honest verifier.

We should emphasize that there is no known attack on the scheme based on making non-random challenges; we are just saying that the proof technique we used previously does not seem to apply in this case. The only known security proofs of the scheme for arbitrary verifiers require additional assumptions.