Work in progress

This is an early draft of a chapter of a book on the foundations of cryptographic authentication being coauthored by Francisco Corella, Sukhi Chuhan and Veronica Wojnas. Please send comments to the authors.

3. Traditional credentials

3.1 Using a key pair for challenge-response authentication

3.2 Public key and attribute certificates

3.3 Hash-of-public-key certificates

3.4 Selective disclosure public key certificates

3.5 From zero knowledge protocols to anonymous credentials

Earlier in this chapter we have seen how *cryptographic credentials* enhance security by authenticating the subject of the credential based on knowledge of a secret that is generated in a cryptographic module and never leaves the module. Then we have seen how *selective disclosure credentials* enhance privacy by only disclosing the user attributes that the relying party needs to know to provide access to resources.

Anonymous credentials further enhance privacy by providing unlinkability of credential presentations. The unlinkability feature of anonymous credentials is typically credited to *zero knowledge protocols*. But how exactly do zero knowledge protocols relate to the unlinkability of anonymous credentials? In this section we shall review a long chain of zero knowledge concepts leading from the original complexity-theoretic concept introduced by the paper for which Goldwasser, Micali and Rackoff received the Turing Award, "The knowledge complexity of interactive proof-systems" [1], to anonymous credentials as they are being specified today by the IRTF using pairings and BBS signatures [2]. At the end of the chain, we shall reach a surprising answer to this question.

3.5.1 Zero knowledge (ZK) proof of a fact (PoF)

The first zero knowledge concept in the chain is *zero knowledge proof of a fact (ZK PoF)* by means of an interactive proof system.

An *interactive proof system* is a pair (P, V) of interactive Turing machines that are configured to execute a protocol where P, the prover, convinces V, the verifier, that a common input s provided to P and V (encoded as a bit string) is an element of a *language* (a set of bit strings) L. The prover is not required to be computationally bounded, but the definition requires V to be efficient, i.e. to run in probabilistic polynomial time. A proof system is said to be *complete* if P has an overwhelming probability (as a function of the length of s) of convincing V that $s \in L$ when that is indeed the case, and *sound* if a cheating prover P' that may not follow the protocol has a negligible probability (also as a function of the length of L) of convincing the verifier V that follows the protocol when the fact is not true.

Informally, an interactive proof system (P, V) for a fact $s \in L$ is *zero knowledge* if a cheating verifier V' verifier cannot efficiently compute anything by interacting with the prover P, which follows the protocol, that V' could not have efficiently computed by itself without interacting with P.

Anything that the cheating verifier V' computes by interacting with P must be derived from the *view* that V' has of the protocol, $\text{View}_{P,V'}(s)$, defined as comprising the messages received from P and the random values that V' generates itself during the interaction, known as its *coin tosses*, including *public coin tosses* generating messages that V' sends to P as specified by the protocol, and *private coin tosses* generating random values that V' uses itself without revealing them to P.

Hence, we can formally say that the protocol is *zero knowledge* if for every interactive Turing machine V' there exists an efficient (i.e. probabilistic polynomial time) Turing machine M, called a *simulator,* such that $\text{View}_{P,V'}(s)$ can be computed as the output $M(s)$ of M on input s; or at least such that $\text{View}_{P,V'}(s)$ and $M(s)$ are *indistinguishable* as families of random variables indexed by $s \in L$.

There are three degrees of indistinguishability between two families of random variables such as $\text{View}_{P,V'}(s)$ and $M(s)$, and three corresponding degrees of zero knowledge. In the above definition, the protocol is said to be:

- *Perfect zero knowledge* if $\text{View}_{P,V'}(s)$ and $M(s)$ are *perfectly indistinguishable,* i.e. *identical.*
- S*tatistical zero knowledge* if $\text{View}_{P,V'}(s)$ and $M(s)$ are *statistically indistinguishable*, which means, by definition, that their *statistical distance*, defined as the sum of the absolute differences $\left|\text{Prob}\left[\text{View}_{P,V'}(s) = \alpha\right] - \text{Prob}[M(s) = \alpha]\right|$ when α ranges over the possible values of the random variables, is negligible as a function of *s.*
- Or *computational zero knowledge* if $\text{View}_{P,V'}(s)$ and $M(s)$ are *computationally indistinguishable*, which means, by definition, that they cannot be distinguished by any efficient algorithm. An efficient algorithm D attempting to distinguish $M(s)$ from $\text{View}_{P,V'}(s)$ is unable to do so, by definition, if the absolute difference between the probability that it outputs 1 when given as input $M(s)$ and the probability that it outputs 1 when given as input $\text{View}_{P,V'}(s)$ is negligible as a function of *s*.

The use of perfect vs. statistical zero knowledge is discussed in the next section, and we shall encounter again the notion of computational indistinguishability in Section 3.5.4.

3.5.1.1 Example: zero knowledge proof of quadratic residuosity

There have many changes to the concept of zero knowledge since it was introduced in 1985. The original paper [1] did not use the word simulator, computational indistinguishability was defined in terms of poly-sized families of circuits, and algorithms were allowed to run in expected polynomial, something that is avoided today [3, §4.3.1.6]. Nevertheless, the paper has an excellent example of a zero knowledge proof of a fact that is very useful for two reasons. First, as a typical example of an interactive proof system with a simple proof of soundness. Second, because the proof of zero knowledge has an error, and the alternative proof that we provide below illustrates a key difference between the first two concepts in the chain of ZK concepts leading to anonymous credentials.

The example that we are referring to is the zero knowledge interactive proof system for quadratic residuosity described in [1, §5]. To facilitate comparison with the paper, in this section we use the letters A and B rather than P and V to refer to the prover and the verifier.

The language QR is defined in [1] as the set of pairs (x, y) where x is a natural number and y is a quadratic residue modulo x. We shall assume that $x > 1$. There are two definitions of the term *quadratic residue*. In a more general definition, a quadratic residue is a positive integer y less than x that is congruent to a perfect square modulo x. A more restrictive definition requires y to be coprime with x, i.e. to be an element of the set $\mathbb{Z}_x^*$ defined in the paper as the set of integers coprime with x, greater than 1, and less than x. A number theoretic result included in the paper without proof as Fact 4, which states that every quadratic residue modulo x has the same number of square roots (that number is 2), is needed for proving zero knowledge and does not hold with the more general definition. So, we shall use the more restrictive definition, as the paper does.

The protocol, which we shall call PQR, takes $(x, y) \in \mathrm{QR}$ as common input and allows A to prove to B that y is a quadratic residue without revealing any modular square root of y. This is achieved by executing the following four-step subprotocol a number of times equal to the bit-length m of x:

1. A generates a random quadratic residue u modulo x and sends it to B.
2. B halts if u is not coprime with x. Otherwise, it generates a random bit b and sends it to A.
3. A generates a random square root w of u modulo x if $b = 0$, or a random square root w of uy modulo x if $b = 1$, and sends w to B.
4. B halts if w is not coprime with x. Otherwise, it verifies that either $b = 0$ and $w^2 = u \bmod x$, or $b = 1$ and $w^2 = uy \bmod x$, and sends an end-of-subprotocol message with the value "reject" if verification fails, "accept" if verification has succeeded m times, or "continue" if it has succeeded less than m times.

Soundless and completeness

The proof system is clearly complete. If A knows that y is a quadratic residue modulo x, it should know a square root of y, and since it generates u in step 1, it should know a square root of u. Hence it should be able to send as w a square root of u or a square root of uy to B in step 3 as required by the value of b received in step 2, and B should be able to use that w in step 4 to verify the appropriate verification equation and accept the proof.

To see that it is sound, consider a cheating prover A' that tries to prove to B that $(x, y) \in$ QR when y is NOT a quadratic residue modulo x.

The u that A' sends to B in step 1 of the subprotocol may or may not be a quadratic residue. If it isn't, the verification in step 4 will fail in the case where $b = 0$, which will occur with probability $\frac{1}{2}$ since B selects b at random. If u is a quadratic residue, then uy is not, and verification will fail in the case where $b = 1$, which will also occur with probability $\frac{1}{2}$. Thus, in all cases, verification of the subprotocol will fail with probability $\frac{1}{2}$. Therefore, verification of the entire sequence of subprotocols can only succeed with probability at most $\frac{1}{2^m}$.

Let l be the bitlength of the common input (x, y). We are assuming that y is not a quadratic residue modulo x, but if it were, it would be less than x. Therefore, B should reject right away if $l > 2m$, so we can assume that $\mathrm{m} \geq \frac{l}{2}$ without loss of generality. Hence the probability that A' will convince B of the false statement that $(x, y) \in$ QR is at most $\frac{1}{2^{(l/2)}}$, which is a negligible function of the length l of the common input (x, y). Therefore, the protocol is sound.

Error in the proof

In the definition of zero knowledge, the view that is indistinguishable from the output of the simulator comprises the coin tosses of the cheating verifier. In [1, §3.4], the paper states that the coin tosses of the verifier B that follows the protocol are an essential part of the view. Later in that section it adds that it is not necessary to include the random bits of the cheating verifier B' in the view (presumably meaning those other than the coin tosses of B specified by the protocol) "because we have included in the view the messages sent by B', and for every B' there is a B'', which is like B' except that it sends its random bits as part of its last message". But no simulator is defined for B'': in the zero knowledge proof of quadratic residuosity of [1, §5], there is no mention of a B'' or of a last message where random bits are sent, and the only coin tosses of the simulator are the random bits of B specified by the protocol.

In fact, it is clearly impossible to prove zero knowledge for a protocol where an arbitrary cheating verifier B' can make arbitrary coin tosses not defined by the protocol without incorporating the unspecified code of B' into the simulator, as is done for example in [3, §4.3.2], or as we do in the following proof.

Our proof of zero knowledge

To prove that PQR is zero knowledge we first need to describe the $\text{View}_{A,B'}(x,y)$ of a cheating verifier B' when interacting with A on the common input (x,y). It is a sequence of m subviews corresponding to the m four-step subprotocols specified above. Each subview comprises, in this order:

- The random quadratic residue u that A sends to B'.
- The bit b that B' sends to A; even though B' is a cheating verifier, in a proof of zero knowledge we can assume without loss of generality that it does send the bit b, because otherwise the sequence will not continue and B' will not gain any more knowledge than it would by not sending the bit. On the other hand, we do not assume that B' generates b at random with uniform distribution over $\{0,1\}$.
- The random square root w of u or random square root w of uy that A sends to B'.
- Zero or more private coins generated by B', not specified by the protocol. Different private coins may be generated in different subviews.

Now we are going to construct a simulator M that takes (x,y) as input and outputs a random variable indistinguishable, in a degree to be determined, from $\text{View}_{A,B'}(x,y)$. M uses its own code to build a model B^* of the cheating verifier B' and a model A^* of an interactive Turing machine that interacts with B^* and, as we shall see below, follows the PQR protocol. A^* and B^* perform a sequence of steps that we shall call the *protocol model* of M, consisting of up to m subsequences, each subsequence comprising the following steps:

1. A^* generates a "tentative" random bit b_t.
2. A^* generates a random positive integer w less than x that is coprime with x.
3. A^* computes $u = w^2 \bmod x$ if $b_t = 0$ or $u = w^2 y^{-1} \bmod x$ if $b_t = 1$.
4. A^* sends u to B^*.
5. B^* sends a bit b to A^*.
6. If $b \neq b_t$ M rewinds A^* and B^* to the beginning of the subsequence, after incrementing a count of consecutive rewinds and terminating with output $\perp$ if the count has reached m.
7. A^* sends w to B^*
8. B^* makes zero or more private coin tosses.
9. B^* sends an end-of-subprotocol message to A^*.
10. M adds the message u of step 4, the bit b of step 5, the message w of step 7, and any private coin tosses made in step 8 to the view of B^* that it is constructing.
11. If the value of the end-of-subprotocol message of step 9 is "reject" or "accept", M terminates and outputs the view of B^* that it has been constructing. Otherwise, the sequence continues at step 1 of the next subsequence.

In Step 3 of the above protocol model:

- If $b_t = 0$, w is a random square root of u modulo x as in Step 3 of the PQR protocol when $b = 0$, and u is a random uniformly distributed quadratic residue modulo x as in step 1 of PQR, because all quadratic residues have the same number of square roots by Fact 4, hence they are all equally likely to be the value of $w^2 \bmod x$.
- If $b_t = 1$, w is a random square root of uy modulo x as in Step 3 of PQR when $b = 1$, and u is again a random uniformly distributed quadratic residue for the same reason as above, plus the fact that multiplying by y^{-1} implements a permutation over $\mathbb{Z}_x^*$.

Hence in Step 4 of the protocol model, A^* sends to B^* a random quadratic residue u modulo x as in Step 1 of the PQR protocol, and in Step 7 of the protocol model A^* sends to B^* a random square root w of u modulo x if $b = 0$, or a random square root w of uy modulo x if $b = 1$. Therefore, A^* implements the prover role of the PQR protocol.

It remains to show that the view of B^* is indistinguishable to some degree from the view of B'. If the target degree is *perfect* indistinguishability, this is actually not true, for three reasons:

1. B^* may halt at any time. In particular, it may halt and not set the bit b in step 5 of the protocol model.
2. B^* may send an unwarranted early reject in step 9 of the protocol model.
3. M may output $\perp$ in step 6 of the protocol model after m consecutive rewinds.

Reasons 1 and 2 may be eliminated by changing the above definition of zero knowledge, adding the condition that "… $\text{View}_{P,V'}(s)$ and $M(s)$ are indistinguishable" *as long as V' does not quit early*. This condition makes a lot of sense, and it makes so much sense that it can be omitted because it goes without saying.

As for the third reason, in [3, Definition 4.3.1] as modified by [3, Definition 4.3.3], Oded Goldreich suggests coping with the possibility that the simulator may output $\perp$ by allowing the output of the simulator to be indistinguishable from the view of the cheating verifier only when the former is conditioned (in the sense of conditional probability) on not being $\perp$. He justifies this relaxed definition by stating in Footnote 7 that he does not know of any non-trivial cases where the non-relaxed definition is satisfied. However, although this may have been true when his textbook [3] was published in 2001, it is no longer true today, since the non-relaxed definition is used for HVZK, as we shall below in Section 3.5.2.

We shall instead deal with the third reason by following a different suggestion also made by Goldreich, in the paragraph that follows [3, Definition 4.3.1]. Since b_t is random in Step 1 of the protocol model, the probability that $b \neq b_t$ in Step 6 is $\frac{1}{2}$, and the probability that M outputs $\perp$ is $\frac{1}{2^m}$, which is negligible in m, and also in the length l of the common input (x, y), which is less than $2m$ because $y < x$. Hence the statistical distance between the view of B^* constructed by M and the view of B' is negligible.

Therefore, the PQR protocol is an interactive proof system that provides *statistical zero knowledge*.

3.5.2 Honest-verifier zero-knowledge (HVZK) proof of knowledge (PoK) of a secret

Proving a fact does not authenticate the prover. To authenticate, the prover must prove knowledge, or synonymously possession, of a secret, as we saw for example in Section 3.1, where the subject of a certificate authenticates by proving knowledge of the private key associated with the public key in the certificate.

The next link in the chain of zero knowledge concepts encompasses protocols that can be used to prove knowledge of a secret but can only be said to be zero-knowledge if the verifier is honest and does not depart from the protocol.

<<< must define HVZK, not just PoK !
An interactive proof system is said to provide *proof of knowledge* of a secret if whenever a prover, which may or may not follow the protocol, convinces the verifier that it knows a secret, an efficient deterministic algorithm, which we shall call a "secret extractor", that can extract the secret from the prover if given full access to the prover including the ability to rewind it.

Honest-verifier zero knowledge (HVZK) is a surprising concept, which at first glance does not seem useful. But if an honest verifier cannot learn anything else from the proof of knowledge other than the fact that the prover knows the secret, neither can eavesdroppers. So HVZK in conjunction with security against direct attack [2, Definition 18.2] implies security against eavesdropping [4, Definition 18.6], as shown in [4, Theorem 19.3]. (We shall discuss various kinds of attacks against authentication, including phishing attacks, man-in-the-middle attacks, and man-in-the-middle phishing attacks in Chapter 4.) And, as we shall see, HVZK is a key link in the chain leading to anonymous credentials.

Honest-verifier proof of knowledge (HVZK PoK) can be implemented by a *Sigma protocol* that provides *special soundness*.

3.5.2.1 Sigma protocols

<<< y instead of s ???
A *Sigma protocol* is an interactive proof system intended to prove a fact $s \in L$ just as in Section 3.5.1, but the language L has an associated binary relation R such that $s \in L$ if and only if there exists w such that $(w, s) \in R$; the prover proves the fact by proving knowledge of a witness, and we are interested in the case where the witness is a unique secret that identifies the user. The name Sigma comes from the shape of a diagram showing the messages exchanged by the prover and the verifier. The prover sends a commitment u to the verifier, the verifier sends a challenge c to the prover, the prover sends a response z to the verifier, and the verifier checks an acceptance condition

relating u, c, z and y. The sequence of messages (u, c, z) is called an *accepting conversation* for y if the acceptance condition is satisfied.

A Sigma protocol provides *special soundness* if there is an efficient deterministic algorithm called a *witness extractor* that outputs a witness from any pair of accepting conversations that start with the same commitment. When a sigma protocol that provides special soundness is used for HVZK PoK, the witness extractor plays the role of what we called above the “secret extractor”.

3.5.2.1 Example: The Schnorr identification protocol

The simplest example of an HVZK PoK protocol is the Schnorr identification protocol [5], where a user authenticates by proving knowledge of a discrete logarithm. (In contexts such as image recognition a sharp distinction is made between authentication and identification, but here we will treat them as synonyms in most contexts.)

The protocol uses a cyclic group G of prime order q with generator g. Each user has a private key $\mathrm{x} \in [1, q)$ and a public key $y = g^x$ that serves as the user’s identity. In Schnorr’s paper, the user is issued a certificate signed by a Key Authentication Center that binds y to information about the user. Schnorr describes the protocol using as G a subgroup of the multiplicative group of the field $\mathbb{Z}_p^*$, but mentions the possible use of other groups, such as elliptic curve groups.

Schnorr’s protocol is a *three-move interactive proof*, a.k.a. a *Sigma protocol* [6], between two interactive Turing machines, the prover P and the verifier V, which perform the following steps:

1. P generates a random integer $\mathrm{r} \in [1, q)$, computes $u = g^r$, and sends u to V.
2. V chooses a random element c from $[0, q)$ and sends c to P.
3. P computes $\mathrm{z} = \mathrm{r} + \mathrm{xc} \pmod{q}$ and sends z to the V.
4. V accepts the proof if $g^z = uy^c$; otherwise, V rejects the proof.

In a Sigma protocol, the message u sent by the prover in step 1 is called the *commitment*, the message c sent by the verifier in step 2 is the *challenge,* the message z sent by the prover in step 3 is the *response*, and the condition checked by the verifier in step 4 is the *acceptance condition*. The sequence of messages (u, c, z) is called an *accepting conversation* if they satisfy the accepting condition.

Completeness (a.k.a. correctness)

If P and V follow the protocol, then $g^z = g^{r+xc} = g^r(g^x)^c = uy^c$ and V accepts the proof.

Special soundness

As we said in Section 3.5.1, an interactive protocol for proving a fact is *sound* if, whenever a prover convinces a verifier that a fact is true, the fact is indeed true. A different definition of soundness is used for a protocol that proves knowledge of a secret. A proof-of-knowledge protocol is said to be *special sound* if, whenever a prover that may or may not follow the protocol convinces the verifier that it knows a secret, an efficient algorithm, called an *extractor*, that is given full access to the prover including the ability to rewind it, can obtain the secret from the prover.

When the proof-of-knowledge protocol is a Sigma protocol, there is a sufficient condition for the existence of an extractor that is sometimes considered the definition of special soundness. A Sigma protocol for proving knowledge of a secret is special sound if the prover's secret can be computed from two accepting conversations having the same commitment but different challenges. An extractor can obtain two such conversations by rewinding the prover after the first conversation to its internal state after sending the commitment; the verifier will then send a second challenge which will be different from the first with high probability.

Schnorr's protocol is a special sound Sigma protocol. Indeed, let the above (u, c, z) be an accepting conversation, and let (u, c', z') be a second accepting conversation with same commitment u. Then we have the two accepting conditions $g^z = uy^c$ and $g^{z'} = uy^{c'}$ and, by dividing the first equation by the second,

$$g^{z-z'} = y^{c-c'}. \tag{1}$$

Since c and c' are distinct elements of the interval $[0, q)$, their difference $c - c'$ is a non-zero element of that interval, and since q is prime, it has an inverse d modulo q. By raising both sides of (1) to d we get

$$g^{(z-z')d} = y^{(c-c')d}. \tag{2}$$

Since q is prime, by Theorem 6, y is of order q like g and both exponents of (2) can be reduced modulo q. Therefore, if we let $w = (z - z')d \bmod q$, we have $g^w = y$ and w is the discrete log of y, which is the secret to be extracted.

Honest-verifier zero-knowledge

An interactive proof-of-knowledge protocol is said to be honest-verifier zero-knowledge if the probability distribution of a random variable denoting the interactions between the prover and the verifier who both follow the protocol is indistinguishable from the probability distribution of a random variable that is output by an efficient Turing machine that simulates the protocol. This is the case for Schnorr's protocol. The random variable denoting the interactions between P and V is a triple of random variables (u, c, z), which can be constructed backwards by the simulator by generating z and c at random, then computing $\mathrm{u} = g^z y^{-c}$. An argument as to why the backwards construction by the simulator results in the same probability distribution as the forward construction by the run of the protocol is sketched in [4, Theorem 19.4].

Security reduction to the discrete log assumption

The special soundness feature of Schnorr's protocol provides a security reduction to the discrete log assumption in the group G where the protocol is to be used. Suppose it were easy for a cheating prover P' who does not know the secret x to convince the verifier. That means it would be easy for P' to reach an accepting conversation. An adversary trying to break the discrete log assumption for the group G could then run P' to obtain an accepting conversation (u, c, z), rewind P', obtain a second conversation (u, c', z'), and if $c \neq c'$, compute the discrete log of u from the two accepting conversations. It would be easy for the DL adversary to obtain the accepting conversations, but c and c' might not be different. The verifier chooses each challenge at random from the interval $[0, q)$, which has 2^q elements, but only some of those elements might be usable by the cheating prover to reach accepting conversations. The security reduction does go through, but the proving it is not trivial. A proof can be found in [4, Theorem 19.1].

3.5.3 Non-interactive zero-knowledge proof of knowledge of a secret

The next link in the chain of zero-knowledge concepts, "non-interactive zero-knowledge proof of knowledge (NIZK PoK)", like the "honest-verifier zero-knowledge" concept of the previous section, is also surprising: how can you prove knowledge of a secret without interaction? You cannot. The protocol that we are going to describe in this section provides the verifier with a proof constructed by a prover who knows a secret but provides no evidence that the knower of the secret is the party at the other end of the communication channel through which the verifier receives the proof. NIZK PoK is also not a zero-knowledge protocol as usually defined, since there is no interaction that can be simulated. But the terminology can be justified by the fact that the one-move protocol of this section is derived from the three-move protocol of the previous section and is used in the construction of anonymous credentials.

Schnorr's identification protocol can be turned into a non-interactive protocol by letting the challenge be provided by the prover rather than the verifier. The prover cannot choose the challenge arbitrarily, because as seen in the discussion of honest-verifier zero knowledge of the previous section, a cheating prover could then choose z and c, then compute $\mathrm{u} = g^z y^{-c}$ and send the accepting conversation (u, c, z) to the verifier.

The prover must be required to compute u first, and this can be enforced by making c dependent on u. One way of making c dependent on u is to let it be the image of u by a function, and a good function to use for this purpose is a cryptographic hash function, which can then be modeled as a random oracle [7] in formal arguments.

So here is a one-move protocol between a prover P and a verifier V derived from Schnorr's three-move protocol. As in Section 3.5.2, G is a cyclic group of prime order q with generator g. P has a private key $\mathrm{x} \in \mathbb{Z}_\mathrm{q}$ and a public key $y = g^x$. H is a cryptographic hash function that takes as argument an element of G and returns an element of $\mathbb{Z}_q$. P and V perform the following two steps:

1. P performs the following computations:
    a. It generates a random integer $\mathrm{r} \in \mathbb{Z}_\mathrm{q}$ and computes $u = g^r$.

   b. It computes $c = H(u)$.
   c. It computes $z = r + xc$.

   Then it sends (c, z) to V as the proof.
2. V computes $u = g^z y^{-c}$ and accepts the proof if $c = H(u)$, otherwise V rejects the proof.

As explained above, this protocol, being non-interactive, cannot be used by P to authenticate to V. If (c, z) were used for authentication as a bearer token and P presented it to V, V could turn around and impersonate P by presenting it to a third party. But the protocol can be modified to be used for authentication by adding interaction to it, as explained in the next section.

3.5.3.1 From NIZK PoK to challenge-response authentication

The one-move protocol of the previous section can be turned into a challenge-response protocol that supports authentication by adding an authentication challenge $c' \in \mathbb{Z}_q$ as a second argument of the call to the hash function $c = H(u, c')$. The original interactive challenge c is then a cryptographic hash, but there is a new challenge c'. The verifier sends the new challenge to the prover before the prover performs its first step, so we now have the following two-move protocol:

1. V chooses a random element $c' \in Z_q$ and sends it to P.
2. P performs the following computations:
   a. It generates a random $r \in \mathbb{Z}_q$ and computes $u = g^r$.
   b. It computes $c = H(u, c')$.
   c. It computes $z = r + xc$.

   Then it sends (c, z) to V.
3. V computes $u = g^z y^{-c}$ and accepts the proof if $c = \mathrm{H}(u, c')$, otherwise V rejects the proof.

NIZK PoK is versatile. We show next how a slightly different modification turns it into a signature scheme.

3.5.3.2 From NIZK PoK to a signature scheme

In the previous section, after turning the Schnorr three-move protocol into a one-move protocol by letting $c = H(u)$, we added a second argument c' to H in $c = H(u, c')$, and used c' as a verifier challenge in a challenge-response protocol. If instead of c' we add a message m as second argument in $c = H(u, m)$, we get instead a *signature scheme*, where P becomes the signer S:

1. S performs the following computations:
   a. It generates a random $r \in \mathbb{Z}_q$ and computes $u = g^r$.
   b. It computes $c = H(u, m)$.
   c. It computes $z = r + xc$.

   Then it sends (c, z) to V as the signature on m.

2. To validate the signature, V computes $u = g^z y^{-c}$ and verifies that $c = H(u, m)$, thus validating the verification equation $\mathrm{c} = \mathrm{H}(g^z y^{-c}, m)$.

3.5.3.3 Two versions of Schnorr signatures

In Chapter 2, Section 2.2.3.2.2, we described the *Schnorr signature scheme*. Here we have just seen how a signature scheme can be derived from the Schnorr identification protocol. Is the above signature scheme the same Schnorr signature scheme that we saw in Chapter 2?

It's hard to tell because we have used different variables in the two schemes. In Chapter 2 we used variable names that were part of a unified notation for comparing signature schemes. Here are using the variable names used in the identification protocol, which are themselves borrowed in part from modern descriptions of the protocol. But it is easy enough to unify the naming, e.g. by changing the variable names *Q, d, R, e, s* of Chapter 2 to *g, x, u, c, z.*

If we do that, the two schemes look very similar, but not identical. In Chapter 2, the equation that binds the public and private keys is $y = g^{-x}$, whereas here it is $y = g^x$. And in Chapter 2, the verification equation is $c = H(g^z y^c, m)$, whereas here it is $c = H(g^z y^{-c}, m)$. The minus sign, so-to-speak, has found its way from the key-pair equation to the verification equation.

The difference is due to the fact that Schnorr's was the first signature scheme based on the discrete log assumption. Today there are many primitives that rely on the discrete log assumption, and it has become customary, as seems natural, to let the private key be the discrete log of the public key in such schemes. That custom had not been established when Schnorr proposed his scheme in 1989 and Schnorr liked it better for his own reasons to put the minus sign in the key-pair equation.

Does this difference matter?

If we replace y with g^{-x} in the verification equation of the Chapter 2 scheme, and with g^x in the verification equation of the Chapter 3 scheme, we get same equation $c = H(g^{z-xc}, m)$ in both cases. So the difference seems cryptographically insignificant. But a party programmed to verify signatures produced by one kind of scheme cannot verify a signature produced by the other. In other words, the two schemes are interoperable.

This illustrates the importance of specifications such as the one standardized by the IETF or the W3C to ensure interoperability.

3.5.3.4 The Fiat-Shamir transform

In Section 3.5.3 we turned Schnorr's three-move identification protocol into a non-interactive, one-move protocol by replacing the challenge sent by the verifier with a cryptographic hash of the commitment. This is a special case of a technique known as the *Fiat-Shamir transform* that is widely used for various purposes in various contexts. In Sections 3.5.3.1 and 3.5.3.2 we saw how it can be used to turn the same three-move protocol into a two-move challenge-response authentication protocol or a signature scheme. In Section 3.5.5.1 we shall see how it can be used to implement unlinkable presentations of anonymous credentials and in Section 3.5.5.2 how it enables signatures by the anonymous subject of a credential. In the original use case [8], Fiat and Shamir used it to turn a sequence of three-move protocols into a single signature by the prover.

In this section we are going to formulate three semi-formal definitions of the Fiat-Shamir at increasing levels of generality.

3.5.3.4.1 Fiat-Shamir transformation of a Sigma identification protocol to a signature scheme

The following definition generalizes the transformation from the Schnorr identification protocol of Section 3.5.2 into the Schnorr signature scheme of Section 3.5.3.2. It can be formulated as follows:

1. In the identification protocol:
    a. The prover sends a commitment, saves the committed value in its internal state, receives a challenge, and sends a response computed from the challenge, the committed value and the prover's private key.
    b. The verifier computes the commitment using a procedure that takes as inputs the challenge, the response and the prover's public key, and checks that the commitment as computed by the procedure agrees with the commitment as received from the prover.
2. In the signature scheme:
    a. The prover performs the same computations as in the identification protocol, using as challenge a hash of the commitment and the message to be signed.
    b. The signature comprises the challenge and the response and can be verified by computing the commitment using the same procedure as in 1b and checking that the challenge is the hash of the commitment and the message.

A formal specification of this transform can be found in [9, Construction 12.9], along with a proof in [9, Theorem 12.10] that the output of the transform is a secure signature scheme if the input is a secure identification protocol. From [9, Theorem 12.11] and the fact the that Schnorr's identification protocol is secure as proved in [9, Theorem 12.10] it follows that Schnorr's signature scheme is secure.

3.5.3.4.2 Fiat-Shamir transformation of a general three-move protocol into a general one-move protocol

The definition in this section covers transformations from a three-move protocol that is not necessarily an identification scheme into a one-move protocol that is not necessarily a signature scheme. The prover has secret data that may consist, for example, of the attributes and issuer's signature of an anonymous credential instead of the prover's private key. The prover has public data that might be the public key of the issuer of an anonymous credential instead of the public key of the prover.

This definition may be formulated as follows:

1. In the three-move protocol:
   a. The prover sends a commitment u, saves the committed value in its internal state, receives a challenge, and sends a response computed from the challenge, the committed value, and the prover's secret data.
   b. The verifier checks the validity of one or more verification equations over the commitment, the challenge, the response and/or the prover's public data and accepts the interactive proof if and only if the equations are satisfied.
2. In the one-move protocol:
   a. The prover performs the same computations as in 1a, using as challenge the result $\mathrm{c} = \mathrm{H}(u, \text{ parameters})$ of calling a cryptographic hash function H on the commitment u and parameters such as a message to be signed or an authentication challenge to be used for challenge-response authentication.
   b. The prover outputs a proof that may comprise the challenge and/or any data items included in the commitment, the response or the prover's public data, and can be verified using the same verification equations as in 1b and the additional equation $c = H(u, \text{ parameters})$.

The one-move protocol resulting from the transformation can be used in one of the following two modes or a combination of both:

A. As a challenge-response authentication protocol, by taking as an input, before the one move, an authentication challenge c' generated by the verifier and passing it as an argument in the call $c = H(u, c')$ of the Fiat-Shamir hash function H, as explained in more detail in Section 3.5.3.1; or
B. As a signature scheme, by passing a message m as an argument in the call $c = H(u, m)$ of the Fiat-Shamir hash function H as explained in Section 3.5.3.2.

We shall see how Mode A can be used for presentation of an anonymous credential in Section 3.5.5.1 and how Mode B can be used to implement message signing by the anonymous user of a credential in Section 3.5.5.2.

3.5.3.4.3 Fiat-Shamir transformation of a sequence of three-move protocols into a single one-move protocol

In the use case where Fiat and Shamir introduced their technique [9] an identification scheme was transformed into a signature. But the identification scheme repeated a three-move protocol t times. In each repetition the verifier sent to the prover as a challenge a random vector of k bits, and the product kt was a security parameter.

To cover this use case we need to define the Fiat-Shamir transform as constructing a non-interactive protocol from an interactive protocol that may consist of one or more three-move subprotocols. Such a definition can be formulated as follows:

1. The interactive protocol consists of one or more three-move subprotocols each performing an interactive subproof, where in each subprotocol:
    a. The prover sends a commitment, saves the committed value in its internal state, receives a challenge, and sends a response computed from the challenge, the committed value, and the prover's secret data.
    b. The verifier checks the validity of one or more verification equations over the commitment, the challenge, the response and/or the prover's public data and accepts the subproof if and only if the equations are satisfied.

    The verifier accepts the overall proof if and only if it accepts all the subproofs.
2. In the non-interactive one-move protocol:
    a. The prover performs the computations that it performs in the steps 1a of all the subprotocols.
    b. The prover computes a challenge string as a cryptographic hash of the commitments of all the subprotocols, and parameters such as a message to be signed, or a challenge to be used for challenge-response authentication, or the attributes of an anonymous credential that the subject wishes to disclose.
    c. The prover outputs a proof comprising data items included in the prover's public data, the challenge string, and the sequence of commitments and responses of the subprotocols.
    d. The proof can be verified using the verification equations of the subproofs, using substrings of the challenge as challenges of the subproofs.
    e. The non-interactive proof is deemed to be valid if and only all the subproofs are valid.

3.5.4 Honest-verifier zero-knowledge proof of knowledge of a signature

We have looked so far at three zero-knowledge concepts: zero-knowledge proof of a fact, honest-verifier zero-knowledge proof of knowledge of a secret, and non-interactive zero-knowledge proof of knowledge of a secret. The fourth concept in the chain leading to anonymous credentials is zero-knowledge proof of knowledge of a signature. We are now getting close to the end of the chain, because as we shall see in the next section, an anonymous credential is essentially a secret signature by the issuer on the attributes of the subject, along with the attributes themselves.

A Sigma protocol can be used to prove knowledge of a secret signature in a way similar to how it can be used to prove knowledge of a private key, with one important difference. The prover sends a commitment, the verifier sends a challenge, the prover sends a response, and the verifier validates verification equations over the commitment, the challenge, the response and a public key. But the public key does not pertain to the prover, which does not have an associated private key. It is associated with the private key used to create the signature, hence it pertains to the signer. This is the important difference. The public key of the signer does not link proofs of knowledge of a signature, whereas the public key of the prover in the identification protocol of Section 3.5.2 links proofs of knowledge of the private key.

We shall go over a detailed example of honest-verifier zero-knowledge proof of knowledge of a signature in Section 3.5.6, where we describe anonymous credentials based on BBS signatures.

3.5.5 Non-interactive zero-knowledge proof of knowledge of a signature

We are now at the end of the chain. Non-interactive zero-knowledge proof of knowledge of a signature is the last zero-knowledge concept needed to understand anonymous credentials.

3.5.5.1 Secret signatures as anonymous credentials

A distinguishing feature of anonymous credentials is that they provide unlinkability of credential presentations. Public key certificates, by contrast, have three elements that are seen by the verifier and may enable tracking: the public key of the subject, the signature by the issuer on the attributes of the subject and the credential metadata, and the attributes and metadata themselves. An anonymous credential eliminates these three sources of linkability.

The public key is eliminated because the subject does not have a public key. The subject does not authenticate by proving knowledge of a private key but by proving knowledge of the credential signature. This requires keeping the signature secret, which in turn eliminates tracking by the signature.

Linkability by attributes that uniquely identify the subject, and the verifier needs to see is unavoidable. But anonymous credentials implement selective disclosure of attributes and metadata items by keeping them secret in a manner similar to how the signature is kept secret, and thus eliminates linkability by the attributes and metadata items that are not disclosed.

There is a difference between the hash-based selective disclosure of attributes in public key certificates that we saw in Section 3.4, and the secrecy-based selective disclosure in anonymous credentials. Hash-based selective disclosure in public key certificates hides attributes behind cryptographic hashes that are covered by the certificate signature and can be used for tracking. Secrecy-based selective disclosure of attributes and metadata in anonymous credentials, on the other hand, eliminates linkability by the attributes and metadata items that are not disclosed by making them part of the secret portion of the credential. It would be possible to use hash-based instead of secrecy-based selective disclosure in anonymous credentials but that would be ill-advised since it would enable tracking unnecessarily.

We shall see an example of secrecy-based selective disclosure in Section 3.5.6.

In Section 3.5.4 we saw how a Sigma protocol can be used to prove knowledge of a secret signature in a manner similar to how Schnorr's identification protocol of Section 3.5.2 proves knowledge of a private key. But Schnorr's protocol is an honest-verifier zero-knowledge protocol, not an unqualified zero-knowledge protocol like the original zero-knowledge protocol of Goldwasser, Micali and Rackoff that we discussed in Section 3.5.1. There is no guarantee that a proof of knowledge of a signature by means of an honest-verifier zero-knowledge protocol will not create tracking opportunities.

This problem can be solved by using the Fiat-Shamir transform to turn the Sigma protocol into a one-move protocol as explained in Section 3.5.3.4.2 and using the one-move protocol in Mode A, i.e. as a challenge-response protocol, by passing an authentication challenge c' generated by the verifier as an argument in the call $c = H(u, c')$. The challenge-response protocol can then serve as the presentation protocol of the anonymous credential.

Recapitulation. An anonymous credential can be implemented as a secret signature by the issuer of the credential on attributes of the subject and credential metadata, some of which may be disclosed by the subject while the others are kept secret along with the signature. The credential can be presented by a challenge-response protocol where the verifier sends an authentication challenge to the subject and the subject responds with a proof of knowledge of the secret signature and the attributes and metadata that are not disclosed, produced by a one-move protocol that is the result of an application to an honest-verifier zero-knowledge proof of knowledge protocol of a Fiat-Shamir transform where the cryptographic hash function takes as an argument the authentication challenge.

3.5.5.2 Signature by the anonymous subject of a credential

In the previous section we saw how the Fiat-Shamir transform can turn an interactive proof of knowledge of the secret signature of an anonymous credential into a one-move protocol, and how the one-move protocol can be used in Mode A as a challenge-response protocol to present the credential. But the one-move protocol can also be used in the Mode B of Section 3.5.3.4.2 as a signature scheme, by passing a message m as an argument in the call $c = H(u, m)$ to the Fiat-Shamir hash function H. The result is a signature on m by the knower of the secret signature of the anonymous credential, i.e. a signature on m by the anonymous subject of the credential.

3.5.6 Example: BBS signatures as anonymous credentials

Boneh, Boyen and Shacham (BBS) signatures [10] are explicitly indended to serve as secret signatures for the implementation of anonymous credentials as explained above in Section 3.5.5.1 and they are currently being standardized by the IRTF for that purpose [2].a

They are based on pairings. A pairing consists of three cyclic groups G_1, G_2 and G_T and a bilinear map $e: G_1 \times G_2 \rightarrow G_T$ that it not degenerate, i.e. that *does not map* generators g_1 and g_2 of G_1 and G_2 to the identity element of G_T. BBS uses G_1, G_2 and G_T of have prime order p, so bilinearity can be defined by $\mathrm{e}(\mathrm{A}^{\mathrm{x}}, \mathrm{B}^{\mathrm{y}}) = \mathrm{e}(\mathrm{A}, \mathrm{B})^{\mathrm{xy}}$ for all $A \in G_1, B \in G_2$ and $x, y \in \mathbb{Z}_p$, from which it follows that $e(AA', B) = e(A, B)e(A', B)$ and $e(A, BB') = e(A, B)e(A, B')$.

There are three types of pairings. In type 1, $G_1 = G_2$. In type 2, $G_1 \neq G_2$ but there is an efficient homomorphism from G_2 to G_1. In type 3, $G_1 \neq G_2$ and there is no efficient homomorphism. BBS is defined on a generic type-3 pairing, and can thus be used with any type of pairing. The IRTF specification uses

- blind issuance
- discuss Tessaro's qualms
- why m' argument to H?
- no NIZK authentication

3.5.7 The role of zero knowledge in anonymous credentials

To conclude this chapter, it may be useful to discuss the role that zero knowledge plays in anonymous credentials.

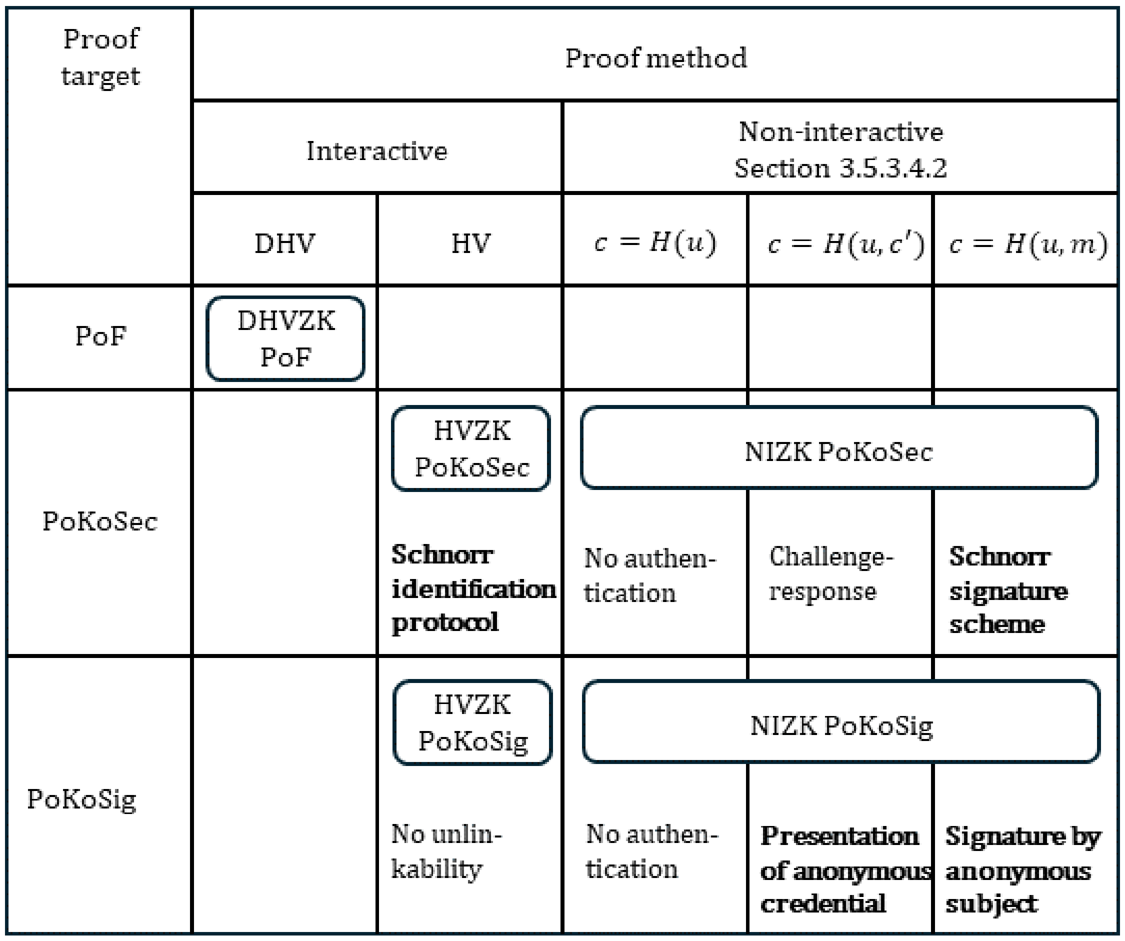
Anonymous credentials are an appealing technology: the subject of an anonymous credential can disclose only the attributes required for a particular presentation; if those attributes are not identifying, the subject remains anonymous based on what is presented; furthermore, presentations are unlinkable, so the subject remains anonymous based on what is presented and how it is presented.

Zero-knowledge proofs are also an appealing technology: they support unlinkability because they allow the prover to prove something without revealing anything else. And anonymous credentials use zero-knowledge technology. That suggests the following causal link: that anonymous credentials allow the subject to remain anonymous because they are presented using zero-knowledge protocols. But this is not so. Paradoxically, even though anonymous credentials use zero-knowledge technology, they are presented using two-move challenge-response protocols.

This paradox is explained by Figure 1, which summarizes Sections 3.5.1-3.5.5. Since an anonymous credential is a secret signature, presentation of anonymous credentials belongs in the last row of the figure. But in that row:

- There is no entry in the DHV column because there are no interactive proof of knowledge protocols guaranteed to be zero-knowledge for dishonest verifiers.
- The next column is the HV column, and HVZK by itself, without Fiat-Shamir transformation, does not guarantee unlinkability.
- The next column is the $c = H(u)$ column, which does not provide authentication for reasons explained in Section 3.5.3.
- The last column refers to signature by the credential subject rather than credential presentation.

That leaves only the $c = H(u, c')$ column for presentation of anonymous credentials, and $c = H(u, c')$ refers to the challenge-response protocol of the Mode A of Section 3.5.3.4.2.



Proof target	Proof method				
	Interactive		Non-interactive Section 3.5.3.4.2		
	DHV	HV	$c = H(u)$	$c = H(u, c')$	$c = H(u, m)$
PoF	DHVZK PoF				
PoKoSec		HVZK PoKoSec	NIZK PoKoSec		
		Schnorr identification protocol	No authen-tication	Challenge-response	**Schnorr signature scheme**
PoKoSig		HVZK PoKoSig	NIZK PoKoSig		
		No unlin-kability	No authen-tication	**Presentation of anonymous credential**	**Signature by anonymous subject**

PoF = Proof of Fact
PoKoSec = Proof of Knowledge of Secret
PoKoSig = Proof of Knowledge of Signature
ZK = Zero-knowledge
NIZK = Non-interactive zero-knowledge
HV = Honest-verifier
DHV = Dishonest-verifier

Figure 1. Zero-knowledge proofs

References

[1] S. Goldwasser et al, "The knowledge complexity of interactive proof-systems," in STOC '85: ACM symposium on Theory of computing, 1985.

[2] T. Looker et al, "IRTF CFRG draft: The BBS Signature Scheme," 12 September 2024. [Online]. Available: https://identity.foundation/bbs-signature/draft-irtf-cfrg-bbs-signatures.html.

[3] O. Goldreich, Foundations of Cryptography, Volume 1 Basic Tools, Cambridge University Press, 2001.

[4] D. Boneh and V. Shoup, “A Graduate Course in Applied Cryptography,” 2024. [Online]. Available: https://toc.cryptobook.us/

[5] C. P. Schnorr, "Efficient Signature Generation by Smart Cards," J. Cryptology, vol. 4, p. 161–174, 1991.

[6] Ivan Damgård, "On Σ-protocols," 2010. [Online]. Available: https://www.cs.au.dk/~ivan/Sigma.pdf.

[7] M. Bellare and P. Rogaway, "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols," in First ACM Conference on Computer and Communications Security, 1993.

[8] A. Fiat and A. Shamir, "How To Prove Yourself: Practical Solutions to Identification and Signature Problems," in CRYPTO'86, 1986.

[9] J Katz and Y. Lindell, Introduction to Modern Cryptography, Second Edition, CRC Press, 2015. S.

[10] S. Tessaro and C. Zhu, "Revisiting BBS Signatures," 2023. [Online]. Available: https://eprint.iacr.org/2023/275.