DIGITAL SIGNATURES, WITH NON-INTERACTIVE PROOFS OF KNOWLEDGE

BURTON ROSENBERG
UNIVERSITY OF MIAMI

Contents

1. Digital Signature

1.1. **General Discussion.** The digital signature is a cryptographic construction that mimics the human ceremony of a signature, such as a signature on a document. Consider the ceremony of signing of a contract.

(1) The contract is written and laid out clearly.
(2) Those that will be committed to the terms of the contract place their hand signature upon the contract.

Date: November 20, 2023/September 6, 2024.

(3) There are witnesses to the signing. For important contracts, the witnesses will be involved explicitly. Even an common place signing, of using a credit card for a coffee, has witnesses.

Our textbook[1] gives three properties of a digital signature, and we look at how these properties exist in the signing ceremony: public verifiability, non-reputability, and transferability.

Public verifiability: The signed contract must be verifiable to parties other than the signers. In disputes might arise which requires adjudication, all parties and verifiers have access to the contract and know the terms of the contract.

Non-reputability: After the conclusion of the ceremony, it must not be easy to disown the contract or the signature. A contract affirms the closure of negotiations and the agreements made. It should not be possible to deny having come to a decision, or the specifics of any of the agreements.

Transferability: The confidence in the signature must be transferable. If there are witness to the signing they can attest to their presence at contract signing, and that the current document is the contract that was signed. The interests of the witness have to be set in suitable opposition to those of the signers so that the witnesses find it advantageous to answer truthfully.

1.2. **Mathematical Definition.**

Definition 1.1. A digital signature is a triple of PPT algorithms,

$$\begin{array}{rrcl} \text{Gen:} & n \in \mathbb{Z} & \mapsto & (pk, sk) \\ \text{Sign:} & m \in \mathcal{M} & \mapsto & \sigma = \sigma_{sk}(m) \\ \text{Verify:} & (m, \sigma) & \mapsto & V_{pk}(m, \sigma) \in \{\, T, F \,\} \end{array}$$

The generation algorithm produces a random n bit a public key, secret key pair. The signing algorithm is given the secret key and a message from the space of messages $\mathcal{M}$ and produces a signature σ of some sort. The verification algorithm is deterministic, and take the public key, the message and the signature and determines if the signature is acceptable.

Correctness: The verifier must accept correct signatures,

$$V_{pk}(m, \sigma_{sk}(m)) = T$$

The verifier accepts a forged signature with less than negligible probability. A PPT adversary,

$$(m, \sigma) \leftarrow_{\Omega} \mathcal{A}^{\mathcal{S}_{sk}}(n, pk)$$

will attempt the forgery. As the notation suggests, the adversary has the public key and access to a signing oracle $\mathcal{S}_{sk}$, It will draw from a distribution of message and

[1]Katz and Lindell *Introduction to Modern Cryptography*, 2-nd ed. page 44.

signatures, with the mission that the signature be valid. A list Q will be maintained of all messages the adversary has inquired of the oracle.

Security:

$$\mathrm{Prob}_\Omega\{\,(m,\sigma) \leftarrow \mathcal{A}^{\mathcal{S}_{sk}} \wedge m \notin Q \wedge V_{pk}(m,\sigma) = T\,\} \leq negl(n).$$

2. Schnorr Identification Protocol

A signature scheme will be described as a modification of an interactive proof system for the secure establishment of identity. An *interactive proof system* is a protocol between a two entities, the *verifier* and the *prover*, two computational devices, where the verifier is constrained to be PPT algorithm. They can share a common input and share a message channel between them, where messages proceed in rounds of query/response between the algorithms.

We will consider the problem of discrete logs. The basis of the problem that modulo an integer, exponentiation is efficient, but the inverse, that of the log to some generator, is exponential time. This is a one-way function in terms of efficiency, and this asymmetry can be used for a cryptographic protocol. One thing we would like to do is see if a prover convince a verifier that it knows the log of a given number in a way that does not reveal what that log is.

A protocol that achieves this aim is called a *zero-knowledge proof of knowlege.*

Definition 2.1. Let G be a group of size q, a large prime with generator g. It can be a subgroup of $\mathbb{Z}_n^*$. Assume that the exponentiation map is one-way. There is a group isomorphism,

$$\begin{array}{rcl} (\mathbb{Z}_q, +) & \xrightarrow{\sim} & (G, \times) \\ i & \mapsto & g^i \pmod G \end{array}$$

Let $\alpha \in_\Omega \mathbb{Z}_q$ be the secret key and $h = g^\alpha \pmod G$ be the public key.
A zero-knowledge proof of knowledge (ZK–PoK) will have three properties,

(1) *correctness:* If the prover has the knowledge, the verifier will be convinced of this. This is also called *completeness.* If the verifier is always convinced, it is called *perfect completeness.*
(2) *soundness:* If the prover does not know the knowledge, it is unlikely the verifier will conclude wrongly that the prover does. Soundness can be demonstrated *knowledge extraction.* For the theory to provide us with benefits, we cannot require that soundness be perfect. The prover must be able to lie its way through this game.
(3) *zero-knowledge*: After the interaction, the verifier has learned nothing except for its conviction on the prover's knowledge of this one fact. This is done with a simulator that can remove the prover and replace it with a PPT algorithm,

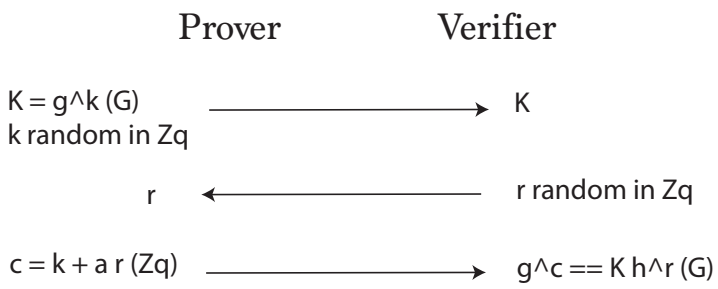

FIGURE 1. The Schnorr Interactive Proof of Knowledge

so that from the perspective of a third party, all computation is consistent with a PPT algorithm without knowledge of the fact.

2.1. **The Schnorr ZK–PoK.** Given the G, β and α as above, the prover proves knowledge of α with one or more rounds of this protocol,

(1) The prover chooses a random $k \in \mathbb{Z}_q$ and commits to it by sending $K = q^k \pmod G$ to the verifier.
(2) The verifier sends a random challenge $r \in \mathbb{Z}_q$ and sends it to the prover.
(3) The prover computes $c = k + \alpha\, r \pmod{\mathbb{Z}_q}$ and sends it to the verifier.
(4) The verifier tests for the equality, $g^c = K\, h^r \pmod G$ and accepts only if the equality holds. Else it rejects.

2.2. **Perfect Completeness.** What the prover is proving to the verifier is that it knows an α such that $h = g^\alpha$. Completeness is the sufficiency of this statement. That is, if the prover knows α then the verifier will accept. This follows from the correctness of the equation,

$$g^c = g^{k+\alpha\, r} = g^k\, g^{\alpha\, r} = g^k\, h^r \pmod G.$$

The completeness is called *perfect* because it the equality will certainly be true.

2.3. **Knowledge Extraction.** We now wish to show necessity. That is, given the protocol, then the prover knows α. What does it mean for an algorithm to know something? In this case it can mean the the algorithm can output the value, or the algorithm can easily infer the value, such as by a simple calculation from other values in has provided.
We ponder the hypothetical case where the prover can receive r and r', two different values, and by perfect correctness it provides correct c and c'. There are two

equations,

$$\begin{aligned} c &= k + \alpha\, r \\ c' &= k + \alpha\, r' \end{aligned}$$

which the prover can solve for α. (We need that $(r - r')^{-1} \pmod{\mathbb{Z}_q}$ exists. It does because q is prime and the difference is non-zero.)
This is called knowledge extraction because we do not know how the prover works, but we do know that it provided the two pairs (r, c) and (r', c'). And that this is sufficient to calculate α.
The point of these protocols is for the prover to prove knowledge without revealing the knowledge. It is in practice highly unlikely that such a pair will occur, because of the large random space and the fair draw from it. With repeated protocol runs, it is an exponentially unlikely event that a certain k is chosen twice. However knowledge extraction only needs to be proven as a possibility, not as an effect procedure.

2.4. **Zero knowledge.** Finally we take up the security of the protocol, which amounts to that α not become known to the verifier, or other PPT algorithm that witnesses the protocol. This is accomplished by noting that the conversation between the prover and the verifier, ignoring the sequence of messages, is a random draw from the space of all triples

$$\mathcal{D}_\alpha = \{\, (K, r, c) \,|\, k, r \in_\Omega \mathbb{Z}_q, K = g^k, c = k + \alpha r \,\}$$

This is not the only way to generate this distribution. In this statement of the distribution, α is required. However, this distribution is stated differently,

$$\mathcal{D}_h = \{\, (K, r, c) \,|\, c, r \in_\Omega \mathbb{Z}_q, K = g^c\, h^{-r} \,\}$$

and $\mathcal{D}_\alpha \cong \mathcal{D}_h$. This second distribution can be created only with public knowledge. Therefore, if one looks at this protocol from the overall computation, the prover can be replaced by a simulator that for each run of the protocol draws a set of messages from $\mathcal{D}_h$, and the protocol completes without knowledge of α.

2.5. **Non-transferability.** One implication is that the conclusion that the verifier, while convinced the prover knows α, cannot convince any third entity that the prover knows α. Because all the verifier can do is produce its logic and transcript, none of which requires α to compute. If knowledge of α were used as if it were a password, this means that the protocol does not allow the verifier to afterwards impersonate the prover. The usual proof of knowledge of a password is to reveal it, and if that were done here the verifier could afterwards impersonate the prover.

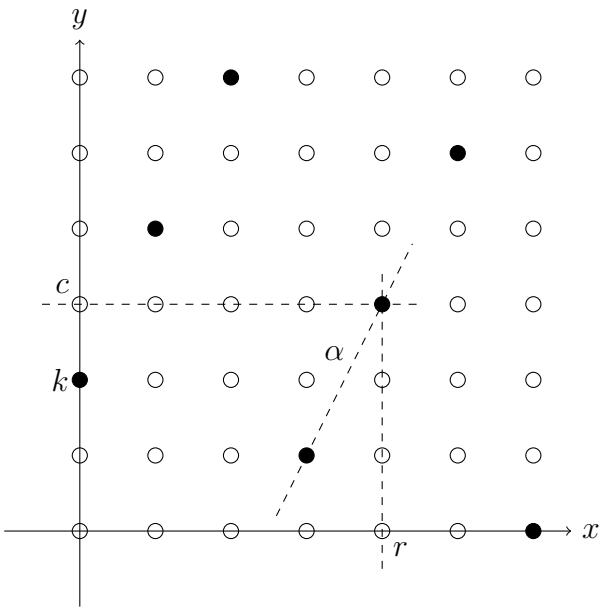

FIGURE 2. $y = k + \alpha\, x$ with challenge r in $\mathbb{Z}_7 \times \mathbb{Z}_7$

2.6. **Obfuscated function evaluation.** The formula in the Schnorr protocol can be compared with that of a line,

$$\begin{aligned} c &= k + \alpha\, r \pmod{\mathbb{Z}_q} \\ y &= b + a\, x \end{aligned}$$

If we identify the secret α with the slope, the verifier is asking for an evaluation of this line at the point of its choosing. We wish to obfuscate the result, so as not to reveal the line's slope, so we choose a random line of slope α. This is done by randomizing the y-intercept. From the equation,

$$k = c - \alpha\, r \pmod{\mathbb{Z}_q}$$

for any c, r pair, there is a k. Hence c is in fact a uniform random choice from $\mathbb{Z}_q$. That is, the math lets us read the equation backwards so that k is dependent, rather than c.

Hence, even though we answer the query correctly, the answer is in essence a random value. What is not random is if if the prover answered two queries for the same line. From this the slope is easily determined, as the y-offset is neutralized in the computation of the slope.

Still, the equation evaluation has to be done in an obfuscated manner, because α and k are not available and will not be revealed. For this reason the equation in $\mathbb{Z}_q$ will be tested in G using the exponential map isomorphism.

2.7. **Paradox.** It is paradoxical that on the one hand we can argue the prover must know α if it can always complete the protocol, but on the other hand a simulator with no knowledge of α can also always complete the protocol. They obvious difference is the order in which the messages are created. In an actual run, the prover commits to k and then is challenged for value of the line $k + \alpha\, r$ for a verifier selected r. Whereas the simulator computes K from c and r, and never does know k.

The paradox is somewhat resolved by considering what if D_h samples randomly (K, r, c) and (K, r', c') (with equal K). Then α is accidentally discovered,

$$g^c\, h^{-r} = g^{c'}\, h^{-r'} \;\Rightarrow\; g^{c-c'}\, h^{r'-r} = 1 \;\Rightarrow\; g^{(c-c')/(r'-r)} = h,$$

so $\alpha = (c - c')/(r' - r) \pmod{\mathbb{Z}_q}$.

We have different ways of understanding the unsoundness of this protocol. Since it is possible that the prover can answer for equal K and unequal r, the prover must know α; but that is not something that is likely to be actually required. So the simulator can keep bluffing its way through until this case occurs, and then it has stumbled upon the value of α.

3. Non-interactive proof of knowledge: Schnorr Signatures

The Schnorr signature scheme is a bottling up of a run of the identification protocol such that no verifier is required. The role of the verifier is to provide a random number in response to K. This can be made non-interactive by using a hash function to choose the r. In the random oracle model, this value is just as random as would be from a true run of the protocol, but is publicly derived from known inputs.

One viewpoint on this is that the hash function is the embodiment of a celestial source of random numbers, which everyone has access to and everyone trusts. The prover uses its own randomness, in the form of K as one input to the hash function, and the other is the message m, so the protocol run is specific for m.

$$\sigma_{sk}(m) = (K, c) \text{ where } r = k + \alpha\, H(m||K)$$

It is also sometimes that the first element of the signature is r rather than K. In this form, the verification is,

$$r := H(m||K)\ ;\ \text{return } g^c == K\, h^r.$$

If in the variant r is provided rather than K, the verification is.

$$K := g^c\, h^{-r} \text{ ; return } r == H(m||K)$$

4. DSA Signatures

The DSA signature is a former NIST standard that modifies the Schnorr equation. This was possibly to avoid the patent on the Schnorr algorithm. The signing equation becomes,

$$\sigma_{sk}(m) = (r, c) \text{ where } c = k^{-1}(H(m) + \alpha\, r),\ r = g^k \bmod G$$

The verification consists of first constructing

$$V = g^{H(m)+\alpha\, r} = g^{H(m)}\, h^r$$

and $c^{-1} \bmod \mathbb{Z}_q$. Then compute,

$$V^{c^{-1}} = (g^{H(m)+\alpha\, r})^{c^{-1}} = g^k \bmod G$$

and check,

$$V^{c^{-1}} == r \bmod G.$$

5. Elliptic Curve Signatures

The Schnorr and DSA algorithms can be used with an elliptic curve public key system. A subgroup $\mathcal{G}$ of q points on the elliptic curve $\mathcal{E}$ is selected, where q is a prime. A point $P \in \mathcal{G}$ is selected. As long as $P \neq 0$, the point generates the group $\mathcal{G}$. The isomorphism that drives the cryptography is,

$$\begin{array}{rcl} (\mathbb{Z}_q, +) & \stackrel{\sim}{\to} & (G, +) \\ i & \mapsto & i\, P \pmod G \end{array}$$

The private key is $\alpha \in_\Omega \mathbb{Z}_q$, and the public key is $Q = \alpha\, P$. The Schnorr signature on message m is, given a random $k \in_\Omega \mathbb{Z}_q$,

$$\sigma_{sk}(m) = (K, c) \text{ where } K = k\, P \text{ and } c = k + \alpha\, H(m||K)$$

The verification is,

$$V_Q(m, (K, c)) \ := \ K + H(m||K)\, Q == c\, P$$

Correctness:

$$\begin{array}{rcl} c\, P & = & (k + \alpha\, H(m||K))\, P \\ & = & k\, P + (\alpha\, H(m||K))\, P \\ & = & k\, P + H(m||K)\, \alpha\, P \\ & = & K + H(m||K)\, Q \end{array}$$

5.1. **Multi-signatures with Schnorr elliptic curve algorithm.** Supposed there are two parties that would like to counter sign a single message m. The parties share the elliptic curve subgroup G, the prime q of the group's order, and a generator P. They each have a secret $\alpha_i \in \mathbb{Z}_q$ and have both published their public keys $Q_i = \alpha_i P \bmod G$.

To sign m, both generate a $k_i \in \mathbb{Z}_q$ and calculate $K_i = k_i P \bmod G$. They share the K_i and form the sum $K = K_1 + K_2$ and the value $r = H(m||K) \bmod \mathbb{Z}_q$. They each calculate,

$$c_i = k_i + \alpha_i r \bmod \mathbb{Z}_q,$$

jointly calculate $c = c_1 + c_2$ and form the signature,

$$\sigma(m) = (r, c) \text{ for public key } Q_1 + Q_2$$

The essence of the verification is,

$$\begin{aligned} c P &= (c_1 + c_2) P \\ &= (k_1 + \alpha_1 r) P + (k_2 + \alpha_2 r) P \\ &= (k_1 + k_2) P + r (\alpha_1 + \alpha_2) P \\ &= (K_1 + K_2) + r (\alpha_1 P + \alpha_2 P) \\ &= K + r (Q_1 + Q_2) \end{aligned}$$

Verification:

$$K = c P - r (Q_1 + Q_2) \text{ ; return } r == H(m||K)$$