Sigma protocol and OR proofs - notes

arnaucube

March 2022

Abstract

This document contains the notes taken during the *Cryptography Seminars* given by Rebekah Mercer.

Contents

1 Sigma protocol **1**
1.1 The protocol 1
1.2 Non interactive protocol 2
1.3 What could go wrong (Simulator) 3

2 OR proof **3**
2.1 The protocol 3
2.1.1 Simulator 3
2.2 Flow 3

3 Resources **4**

1 Sigma protocol

1.1 The protocol

Let q be a prime, q a prime divisor in $p-1$, and g and element of order q in $\mathbb{Z}_p^a$. Then we have $G = \langle g \rangle$.
We assume that computationally for a given A it's hard to find $a \in \mathbb{F}$ such that $A = g^a$.
Alice wants to prove that knows the *witness* $w \in \mathbb{F}$, such that the *statement* $X = g^w$, without revealing w.

1. Alice generates a random $a \overset{r}{\leftarrow} \mathbb{F}$, and computes $A = g^a$. And sends A to Bob.

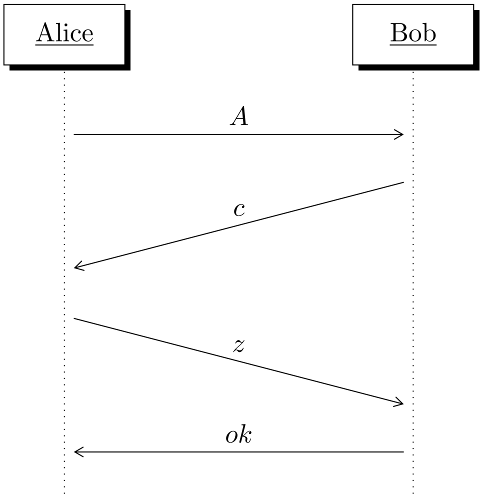
2. Bob generates a challenge $c \overset{r}{\leftarrow} \mathbb{F}$, and sends it to Alice.

3. Alice computes $z = a + c \cdot w$, and sends it to Bob.

4. Bob verifies it by checking that $g^z == X^c \cdot A$.

   We can unfold Bob's verification and see that:

$$g^z == X^c \cdot A$$
$$g^{a+cw} == g^{wc} g^a$$
$$g^{a+cw} == g^{wc+a}$$



   Properties:

i. *correctness/completness*: if Alice know the witness for the statement, then they can create a valid proof.

ii. *soundness*: if someone does not have knowledge of the witness, can not form a valid proof (verifier will always reject).

iii. *zero knowledge*: nobody gains knowledge of anything new with the proof. prior knowledge + proof = prior knowledge

1.2 Non interactive protocol

With the *Fiat-Shamir Heuristic*, we model a hash function as a random oracle, thus we can replace Bob's role by a hash function in order to obtain the challenge $c \in \mathbb{F}$.
So, we replace the step 2 from the described protocol by $c = H(X||A)$ (where H is a known hash function).

1.3 What could go wrong (Simulator)

If the verifier (Bob) sends $c \in \mathbb{F}$, prior to the prover committed to A, the prover could create a proof about a public key which they don't know w.

1. Bob sends $c \overset{r}{\leftarrow} \mathbb{F}$ to Alice
2. Alice generates $z \overset{r}{\leftarrow} \mathbb{F}$
3. Alice then computes $A = g^z X^{-c}$, and sends z, A to Bob
4. Bob would check that $g^z == X^c A$ and it would pass the verification, as $g^z == X^c \cdot A \Rightarrow g^z == X^c \cdot g^z X^{-c} \Rightarrow g^z == g^z$.

As we've seen, it's really important the order of the steps, so Alice must commit to A before knowing c.
This 'fake' proof generation is often called the *simulator* and used for further constructions.

2 OR proof

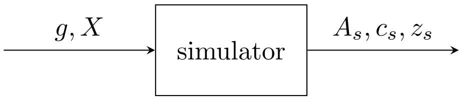
OR proofs allows the prover to prove that they know the witness w of one of the two known *public keys* $X_0, X_1 \in \mathbb{F}$, without revealing which one. It uses the construction seen in the *sigma protocols* together with the idea of the *simulator*.

A similar construction is used for n statements in the *ring signatures* scheme (used for example in *Monero*). In our case, we will work with $n = 2$.

2.1 The protocol

2.1.1 Simulator

We can assume that the simulator is a box that for given the inputs (g, X), it will output (A_s, c_s, z_s), such that verification succeeds $(g^{z_s} == X^{c_s} \cdot A_s)$.



Internally, the simulator computes

$$z_s \overset{r}{\leftarrow} \mathbb{F},\ c_s \overset{r}{\leftarrow} \mathbb{F},\ A_s = g^{z_s} \cdot X^{c_s}$$

2.2 Flow

For two known *public keys* $X_0, X_1 \in G$, Alice knows $w_b \in \mathbb{F}$, for $b \in \{0, 1\}$, such that $g^{w_b} = X_0$ or $g^{w_b} = X_1$. As we don't know if Alice controls 0 or 1, from now on, we will use b and $1 - b$.
So, Alice knows $w_b \in \mathbb{F}$ such that $X_b = g^{w_b}$, and does not know w_{1-b} for $X_{1-b} = g^{w_{1-b}}$.

1. First of all, as in the *Sigma protocol*, Alice generates a random *commitment* $a_b \overset{r}{\leftarrow} \mathbb{F}$, and computes $A_b = g^{a_b}$.

2. Then, Alice will run the *simulator* for $1 - b$.

   Sets a random $c_{1-b} \overset{r}{\leftarrow} \mathbb{F}$, and runs the simulator with inputs (c_{1-b}, X_{1-b}), and outputs $(A_{1-b}, c_{1-b}, z_{1-b})$.

   Remember that internally the *simulator* will set random $z_{1-b}, c_{1-b} \overset{r}{\leftarrow} \mathbb{F}$, and compute an A_{1-b} such that $A_{1-b} = g^{z_{1-b}} \cdot X_{1-b}^{c_{1-b}}$.

3. Now, Alice sends A_b, A_{1-b} to Bob

4. And Bob sends back the *challenge* $s \overset{r}{\leftarrow} \mathbb{F}$.

5. Alice then splits the challenge s into c_b, c_{1-b}, by $s = c_{1-b} \oplus c_b$. So Alice can compute $c_b = s \oplus c_{1-b}$.

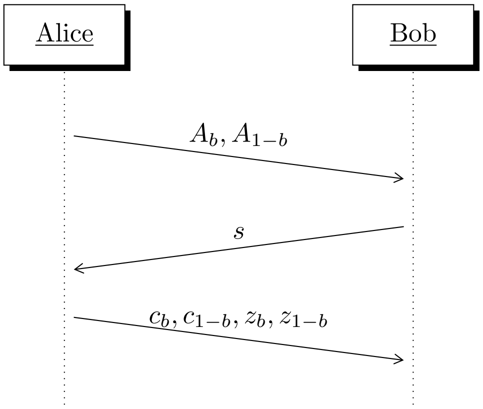
6. Then Alice computes $z_b = a_b \cdot w_b + c_b$. And sends to Bob $(c_b, c_{1-b}, z_b, z_{1-b})$.

7. Bob can perform the verification by checking that:

   i. $s == c_b \oplus c_{1-b}$

   ii. $g_{z_{1-b}} == A_{1-b} \cdot X_{1-b}^{-c_{1-b}}$

   iii. $g_{z_b} == A_b \cdot X_b^{-c_b}$



3 Resources

1. https://cs.au.dk/ ivan/Sigma.pdf

2. *Cryptography Made Simple*, Nigel Smart. Section 21.3.