Introduction to cryptology (GBIN8U16)

✧

More on discrete-logarithm based schemes

Pierre Karpman
pierre.karpman@univ-grenoble-alpes.fr
https://www-ljk.imag.fr/membres/Pierre.Karpman/tea.html

2018–03–13

Signatures: what?

Objectives of a signature algorithm:

- Given $(\mathrm{sk}, \mathrm{pk})$ a key pair
- message m + secret key $\mathrm{sk} \leadsto$ signature $s = \mathfrak{S}_{\mathrm{sk}}(m)$
- message m + signature s + public key $\mathrm{pk} \leadsto$ verified message $\mathfrak{V}_{\mathrm{pk}}(m, s)$

Informal security objectives

- Given pk, it should be hard to find sk
- Given pk, it should be hard to forge signatures
- (Variant: given access to a signing oracle $\mathfrak{O}_{(\mathrm{sk},\mathrm{pk})}$, it should be hard to forge signatures)

Related: interactive proof of identity

Objective of a proof of ID scheme:

- Publish public identification data α
- When challenged, prove knowledge of a secret related to α

Example of a one-time scheme:

1. Let $\mathcal{H}$ be a preimage-resistant hash function, $\mathcal{R}$ a large set
2. The prover draws $x \xleftarrow{\$} \mathcal{R}$, computes and publishes $X = \mathcal{H}(x)$
3. When challenged, reveals x

Many-time variant:

1. Draw $x \xleftarrow{\$} \mathcal{R}$, compute and publish $X = \mathcal{H}^N(x)$
2. When challenged, reveal $\mathcal{H}^{N-1}(x)$, reset $X = \mathcal{H}^{N-1}(x)$

A discrete-log based PoID scheme

From last week's TD (~Schnorr):

1. Let $\mathbb{G} = \langle g \rangle$ be a group with a hard DLP
2. The prover draws $x \xleftarrow{\$} \mathcal{R}$, computes and publishes $X = g^x$
3. When challenged; draws r, sends $R = g^r$
4. The verifier picks c and sends it
5. The prover computes $a = r + cx$ and sends it
6. The verifier checks that $RX^c = g^a$

This can be run many times, BUT r's should be random and never repeat!

From PoID to signature

Differences between PoID and signatures:

- PoIDs are interactive (in the verification), signatures are not
- Signatures also involve a message

One major observation:

- If the prover can convince that it doesn't control *both* R and c, interaction is unnecessary
- (Otherwise, nothing is proved)

⇒ Fiat-Shamir transformation: generate c from R with a hash function

Schnorr signatures

To sign a message m with the key $(\mathrm{sk}, \mathrm{pk})$ pair $(x, X = g^x)$

1. Pick $r \xleftarrow{\$} \mathcal{R}$ and compute $R = g^r$
2. Compute $c = \mathcal{H}(R, m)$
3. Compute $a = r + cx$ and output (c, a) as the signature of m

To verify a signature:

1. Compute $\hat{R} = g^a / X^c = g^a / g^{cx}$
2. Check that $c = \mathcal{H}(\hat{R}, m)$

Important: r must (again) be random and not repeat! (Why?)

Remember randomness (always)!

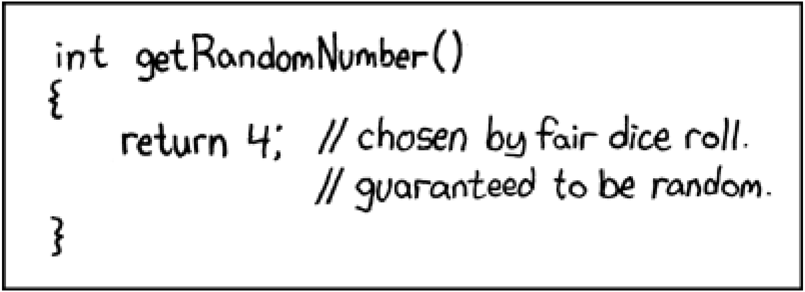

Figure: Not good for Schnorr signatures

Where are we with dlog?

If $\mathbb{G} = \langle g \rangle$ is a prime-order group where the DLP is hard (on average ≡ in the worst case (cf. TD)), then:

- Can do asymmetric key exchange
- Can do public-key signatures

For signatures we also need

- Good hash functions
- Good pseudorandom number generation

Some comments on dlog attacks

When $\mathbb{G} \approx \mathbb{F}_p^*$, the current dlog records are:

- $|p| \approx 768$ bits (Kleinjung et al., 2017), using a *Number Field Sieve* (NFS) algorithm
  - Took about 5300 core years
- $|p| \approx 1024$ bits for a *trapdoored* prime (Fried et al., 2017), using a *Special NFS* (SNFS) algorithm
  - Took about 385 core years

Note: it may be hard to decide if a prime is trapdoored

One nice (for an attacker) feature of (S)NFS:

- The largest part of the cost is a *precomputation*, then computing *individual dlogs* is *very fast*

Some more comments on dlog: small subgroup attack

Consider a *semi-static* key exchange,

- Where one of g^a or g^b (say g^b) is fixed

using $\langle g \rangle \subset \mathbb{F}_p^*$ where $\mathbb{F}_p^*$ has many small subgroups

- Then B must check that "$\hat{g}$" sent by A is in the correct group
- Otherwise, if $\hat{g}^b$ is in a small group of order N, a malicious A can learn $b \mod N$
- ... Then $b \mod N'$, etc.

One way to easily prevent this: use $p = 2q + 1$, q a Sophie Germain prime

$\Rightarrow$ Only a small subgroup of order 2 to check for in $\mathbb{F}_p^*$

What about implementation, though?

- We need to compute g^x, for a large x (e.g. 256 bits)
- Cannot just do $g \times g \times g \times \ldots \times g \approx 2^{256}$ times!
- Notice that $g \times g = g^2$, $g^2 \times g^2 = g^4$, $g^4 \times g^4 = g^{16}$, etc.
- Also: $g \times g^2 = g^3$, $g^2 \times g^{16} = g^{18}$, etc.

$\rightsquigarrow$ "Square & multiply" algorithm

Square & multiply

Square & multiply

Input x, g
Output g^x

```
h = 1
While x ≠ 0
        if (x&1)
                h ↤ h × g
        g ↤ g × g
        x ↤ x ≫ 1
Return h
```

$\Rightarrow$ Only $\log(x)$ iterations needed!
(Problem here, runtime also depends on $\mathrm{wt}(x)$)

Implementation: what else?

- We also need multiplication, addition in $\mathbb{G}$
- If $\mathbb{G} \subseteq \mathbb{F}_p^*$ $\Rightarrow$ modular arithmetic
- Require big number multiplication, (integer) division, remainders, addition
- $\Rightarrow$ split f as e.g. $f_0 + 2^{64} f_1 + 2^{128} f_2 + \ldots$
- Can use dedicated arithmetic for "efficient" primes

Implementation digression

Consider $p = 2^{111} - 37$, then

- $2^{111} \equiv 37 \mod p$
- $a \times 2^{111} \equiv a \times 37 \mod p$
- $a \times 2^{112} \equiv a \times 74 \mod p$
- $a \times 2^{28} \times b \times 2^{84} \equiv ab \times 74 \mod p$
- $a \times 2^{56} \times b \times 2^{56} \equiv ab \times 74 \mod p$
- etc.

Multiplication $\mod 2^{111} - 37$

Let $f = f_0 + 2^{28} f_1 + 2^{56} f_2 + 2^{84} f_3$, $g = g_0 + 2^{28} g_1 + 2^{56} g_2 + 2^{84} g_3$, set

- $h_0 = f_0 g_0 + 74 f_1 g_3 + 74 f_2 g_2 + 74 f_3 g_1$
- $h_1 = f_0 g_1 + f_1 g_0 + 74 f_2 g_3 + 74 f_3 g_2$
- $h_2 = f_0 g_2 + f_1 g_1 + f_2 g_0 + 74 f_3 g_3$
- $h_3 = f_0 g_3 + f_1 g_2 + f_2 g_1 + f_3 g_0$

Then $fg \mod 2^{111} - 37 = h_0 + 2^{28} h_1 + 2^{56} h_2 + 2^{84} h_3 \mod 2^{111} - 37$

To be complete:

- Have to reduce the h_i's $\mod 2^{28}$ (2^{27})
- Have to ensure that all $f_i g_j$ can be computed with, say, a $64 \times 64 \to 64$ multiplier (in fact, desktop CPUs have $64 \times 64 \to 128$ multipliers)

$\Rightarrow$ Modular multiplication w/o explicit division

What next?

In two weeks:

- Inversion in integer rings: extended Euclid algorithm
- The Chinese Remainder Theorem (CRT)
- How to do asymmetric key exchange, public key signatures differently: RSA