Digital signatures

Using multiplicative groups

Algebraic group

Consists of a set $\mathbb{G}$ and a binary operation $\cdot$ with:

- **Closure**: $\forall\ A, B \in \mathbb{G}: A \cdot B \in \mathbb{G}$
- **Associativity** (allows omitting parentheses): $\forall\ A, B, C \in \mathbb{G}: (A \cdot B) \cdot C = A \cdot (B \cdot C)$
- **Identity**: $\exists\ I \in \mathbb{G}\ \forall\ A \in \mathbb{G}: I \cdot A = A \cdot I = A$
- **Invertibility**: $\forall\ A \in \mathbb{G}\ \exists\ B \in \mathbb{G}: A \cdot B = I$

Not required but fulfilled by multiplicative groups:

- **Commutativity**: $\forall\ A, B \in \mathbb{G}: A \cdot B = B \cdot A$

Order of the group and of elements

- **Group order**: number of elements in the set $\mathbb{G}$.
- **Element order**: how many times the element has to be repeated to get the identity element.
- **Multiplicative notation**: $A^n = I$

If $|A| < |\mathbb{G}|$, then A generates a subgroup.

Lagrange's theorem: $\forall\ A \in \mathbb{G}$: $|A|$ divides $|\mathbb{G}|$.

As a consequence, $A^{|\mathbb{G}|} = I$, which is called Fermat's little theorem or Euler's theorem.

Group generator

$G \in \mathbb{G}$ generates the whole group if $|G| = |\mathbb{G}|$ and thus $\mathbb{G} = \{G^i$ with i from 1 to $|\mathbb{G}|\}$.

Repeating a generator is linear: $G^{a+b} = G^a \cdot G^b$.

I use lowercase letters for integers and uppercase letters for elements of the group.

Repetition ring

A ring is a field in which not all elements have a multiplicative inverse (only those which have no factor in common with the ring modulus).

Whether two numbers are coprime can be determined with the Euclidean algorithm.

The multiplicative inverse can be determined with the extended Euclidean algorithm.

Since $G^{|G|} = I$, the coefficients (or exponents) can be calculated modulo $|G|$.

Discrete logarithm problem (DLP)

Discrete logarithm problem: Given $A = G^a$, no efficient algorithms are known to compute a from A and G in multiplicative groups, which makes it a one-way function.

Note that the "discrete root problem" (solving for the base rather than the exponent) is not always difficult.

Digital signature scheme

Digital signature schemes consist of 3 algorithms:

- **KeyGeneration**(entropy) → private k, public K (called key because you can unlock things like coins; k → K typically easy, K → k always hard)
- **Signing**(message, k) → signature (can only be produced by the person who knows the key k)
- **Verification**(message, K, signature) → true/false (anyone who knows the public key K can verify)

Computations with secret values

The idea behind many signature schemes is to use a linear one-way function to hide the private key while still allowing the verifier to compute with it.

Example: You can compute f(a · b) if you know f(b) without having to know b (see Diffie-Hellman).

For all the signature schemes discussed today, k is the private key and $K = G^k$ the public key.

A signature value s has to depend on the private key k and h = hash(message).

An unsuccessful first attempt

- **Equation** (to be masked): $h =_{|G|} k \cdot s$
- **Signing**: $s =_{|G|} k^{-1} \cdot h$
- **Verification**: $G^h \stackrel{?}{=} K^s$

Problem: The verifier can compute $k =_{|G|} h \cdot s^{-1}$.

(One equation with one unknown can be solved.)

Solution: Add a second unknown to the equation.

Ephemeral key for hiding static key

- **Ephemeral key**: private r, public $R = G^r$ (public means R is included in the signature)
- **Equation**: $h =_{|G|} k \cdot s + r$
- **Signing**: $s =_{|G|} k^{-1} \cdot (h - r) \cdot$
- **Verification**: $G^h \stackrel{?}{=} K^s \cdot R$

Problem: Anyone can forge a valid signature by choosing a random s and computing $R = G^h / K^s$.

Moving s to R ($G^h \stackrel{?}{=} K \cdot R^s$) doesn't work because the "discrete root problem" is not always difficult.

Solution: Make s depend on R

There are two ways to do this:

- Use R as an integer in the equation which is solved for s (see the Elgamal signature scheme).
- Include R in the hash: h = hash(message, R) (see the Schnorr signature scheme later on).

Elgamal signature scheme

- **Equation**: $h =_{|G|} k \cdot R + r \cdot s$
- **Signing**: $s =_{|G|} r^{-1} \cdot (h - k \cdot R)$
- **Verification**: $G^h \stackrel{?}{=} K^R \cdot R^s$

Important: The ephemeral key $R = G^r$ has to be different for every signature! Otherwise, the two equations can be solved for the two unknowns.

Described by Elgamal (the father of SSL) in 1985.

Zero-knowledge proofs

Goal: Convince another party of one's knowledge w/o revealing any information or leaving evidence.

Parties: Prover convinces verifier of knowing k. Trivial by revealing k but not zero-knowledge then.

- **Completeness** (successful proof): An honest verifier will be convinced by an honest prover.
- **Soundness** (proof of knowledge): Prover can fake knowledge only with negligible probability.
- **Zero-knowledge**: Verifier can fake transcript.

Knowledge of discrete logarithm

Prover		**Verifier**
knows k so that $K = G^k$		
choose random $r < \|G\|$		
compute $R = G^r$		
	R →	
		choose random $c < \|G\|$
	← c	
compute $s =_{\|G\|} r - k \cdot c$		
	s →	
		verify that $R = G^s \cdot K^c$

Evaluation of criteria

- **Completeness**: $G^r = G^{r - k \cdot c} \cdot (G^k)^c$.
- **Soundness**: By sending distinct challenges c and c' after the same R, the verifier can extract the secret k by computing (s – s')/(c' – c) because $G^s \cdot K^c = R = G^{s'} \cdot K^{c'}$ and thus $G^s / G^{s'} = G^{s - s'} = K^{c'} / K^c = K^{c' - c}$.
- **Zero-Knowledge**: By choosing the random values c and s first, the verifier can compute $R = G^s \cdot K^c$, which results in a valid transcript.

Choice of the challenge

Since the verifier might choose the challenge c non-randomly, this protocol is only so-called **honest-verifier** zero-knowledge. A dishonest verifier can turn this into a signature scheme. (The verifier could commit to c beforehand.)

To be a proof of knowledge, the prover has to learn the challenge c **after** fixing the ephemeral key R. This dependency can be established either by a verifier or by a cryptographic hash function.

Schnorr signature scheme

- **Signer**:
  - Knows k so that $K = G^k$.
  - Choose random $r < |G|$.
  - Compute $R = G^r$, $c = \text{hash}(R, m, K)$, and $s =_{|G|} r - k \cdot c$ for message m.
  - Share (c, s) or (R, s) as a signature.
- **Verifier**:
  - In the case of (c, s): $c \stackrel{?}{=} \text{hash}(G^s \cdot K^c, m, K)$.
  - In the case of (R, s): $R \stackrel{?}{=} G^s \cdot K^{\text{hash}(R, m, K)}$.