CRYPTOGRAPHY LECTURE, 2023

Master level

Mirosław Kutyłowski

Signatures - Proofs of possesion of private key

Concept:

- Alice holds a secret key sk
- nobody else has sk
- there is pk corresponding to sk
- with pk one can check that some data have been generated with sk

$\Rightarrow$ **Proof of Posession** of secret key corresponding to pk

Proofs of Possession

- **interactive:** for **authentication**

  challenge-response algorithms

- **non-interactive**: for **digital signatures**
  - option 1: signature is the result $F(M, \mathrm{sk})$ for some function F
  - option 2: challenge response algorithm simulated - randomness comes from $\mathrm{Hash}(M)$

Security options:

- standard model
- Random Oracle Model - signature may be insecure if Hash turns out to be weak

Signature

signature of M as a proof of posession:

- the signer has used sk
- the proof corresponds to M

Procedures:

- key generation
- signature creation
- signature verification

(signature validation is a broader concept including verification)

Properties

soundness: if $s := \mathrm{Sign}_{\mathrm{sk}}(M)$, then $\mathrm{Verify}(s, M, \mathrm{pk}) = \mathrm{valid}$

if $s' \neq \mathrm{Sign}_{\mathrm{sk}}(M)$, then $\mathrm{Verify}(s', M, \mathrm{pk}) = \mathrm{invalid}$ except for a negligible probability

unforgeability:

- black-box creating signatures on demand of the adversary for messages of his choice
- adversary wins if presents a valid signature that was not the output of the blackbox

some leakage of secret key possible – as far as unforgeability still holds

Schemes with appendix:

components: one-way function F, message encoding μ

signing: $s := F(\mu(M), \mathrm{sk})$

verification: $\mathrm{Verify}(s, \mu(M), \mathrm{pk}) = ?$

Crucial issue: embedding μ

Signature with message recovery

components: one-way function F, message encoding μ

signing: $s := F(\mu(M), A, \mathrm{sk})$

verification: $(\mathrm{status}, M) := \mathrm{Recovery}(s, A, \mathrm{pk})$ where $\mathrm{status} \in \{\mathrm{valid}, \mathrm{invalid}\}$

RSA signatures – basic form

i. $h := \mu(M)$

ii. $s := h^d \bmod n$ for the secret d

iii. output s

textbook version: $\mu = \mathrm{Hash}$

Multiplicative properties:

$$\mu(M) = \prod \mu(M_i)^{\alpha_i}$$

implies that

$$s(M) = s(M_i)^{\alpha_i} \bmod n$$

Attack: if μ maps to integers with small factors, then finding such α's is doable

(e.g. if images of μ are relatively small)

$\Rightarrow$ existential forgery possible no matter how "random" is μ

such RSA might be insecure!

Examples:

ANSI X9.31: $\mu(M) = 6 \| \text{bb....ab} \| \ \text{Hash}(M) \| \text{HashID}$

PKCS#1v1.5: $\mu(M) = 00 \| 01 \| \text{PS} \| 00 \| \text{HashID} \| \text{Hash}(M)$ where PS is a string of FF octets

wrong padding: $\mu(M) = a \cdot M + b$ (for short messages, if we append M with fixed bytes)

RSA with message recovery

ISO9796-1 RSA (M of at most half length of length of n) - completely broken

ISO9796-2 RSA:

signing:

i. $M = m_L \| m_R$

ii. $P := 6A \| m_L \| \mathrm{Hash}(M) \| \mathrm{BC}$

iii. $s := P^d \bmod n$

iv. output s, m_R

verification:

i. $P := s^e \bmod n$

ii. check format of P, derive m_L

iii. recompute $\mathrm{Hash}(M)$ and compare with P

Boneh-Boyen signature scheme -secure in the standard model

setup: bilinear map $e: G_1 \times G_2 \to G_T$, generators g_1, g_2 of G_1, G_2, cyclic with order p

key generation: choose x, y at random, $\zeta = e(g_1, g_2)$,

$$\mathrm{pk} = (g_1, g_2, g_2^x, g_2^y, \zeta), \qquad \mathrm{sk} = (x, y)$$

signing message m:

i. choose r at random such that $x + m + y \cdot r \neq 0 \bmod p$,

ii. $\sigma := g_1^{1/(x+m+y\cdot r)}$

iii. output (σ, r)

verification:

signature valid iff $e(\sigma, g_2^x \cdot g_2^m \cdot (g_2^y)^r) = \zeta$

let us check:

$$e(\sigma, g_2^x \cdot g_2^m \cdot (g_2^y)^r) = e\big(g_1^{1/(x+m+y\cdot r)}, g_2^{x+m+y\cdot r}\big) = e(g_1, g_2)^{x+m+y\cdot r/(x+m+y\cdot r)} = e(g_1, g_2) = \zeta$$

security assumption $q-\mathrm{SDH}$

for bilinear groups:

given $(g_1, g_1^x, ..., g_1^{x^q}, g_2, g_2^x)$ compute a pair $\left(c, {g_1}^{1/(x+c)}\right)$ for any c

let us show that forgery of the signature implies breaking $q-\mathrm{SDH}$:

- for a simpler signature where $\sigma = g_1^{1/(m+x)}$
- **proof for model where:**

  messages to be signed are created before public key is given
- **2nd part of the proof (omitted):**

  Boneh-Boyen signatures can be forged $\Rightarrow$

  simplified signatures can be forged in the above model

  (this is the reason why we need y in the scheme)

reduction: forgery $\Rightarrow$ breaking $q-\mathrm{SDH}$

given instance of $q-\mathrm{SDH}$: $(g_1, d_1, ..., d_q, g_2, h)$

- choose $m_1, ..., m_q$
- define $f(X) = \prod_{i=1}^{q} (X - m_i)$, expand to $f(X) = \sum \alpha_i \cdot X^i$
- take random θ
- set $g_1' := \prod d_i^{\alpha_i \cdot \theta}$ (so $g_1' = g_1^{f(x) \cdot \theta}$) , g_1' uniformly distributed, independent of g_2
- set $\zeta' := e(g_1', g_2)$
- set $\mathrm{pk} = (g_1', g_2, h, \zeta')$
- simulate signatures for m_i:

  a) define $f_i(X) = f(X)/(X - m_i) = \prod_{j \neq i} (X - m_j)$

  b) expand $f_i(X) = \sum \beta_j \cdot X^j$

  c) $\sigma_i := \prod d_j^{\beta_i \cdot \theta}$ so $\sigma_i = {g_1'}^{1/(m_i + x)}$ – signature!

Now use the algorithm to create a forged signature σ_* for m_*

- note that: $\sigma_* = {g_1'}^{1/(m_*+x)} = {g_1}^{\theta \cdot f(m_*)/(m_*+x)}$
- express f in the following way: $f(X) = (m_* + X) \cdot \gamma(X) + \gamma_*$
- $f(X)/(m_* + X) = \gamma(X) + \gamma_*/(m_* + X)$
- $\sigma_*^{1/\theta} = g_1^{\gamma(X)} \cdot g_1^{\gamma_*/(m_*+x)}$
- expand $\gamma(X)$ and compute $g_1^{\gamma(X)}$
- finally solve for $g_1^{1/(m_*+x)}$

  q-SDH solved!

From restricted model to security of full signature

- assume that there is a forger $\mathcal{A}$ for the full scheme, we are going to create a forger $\mathcal{B}$ for a restricted model/signature
- $\mathcal{B}$ creates $w_1, \ldots, w_k$ at random and in response gets
  - a public key (g_1, g_2, u, z)
  - signatures $\sigma_1, \ldots, \sigma_k$
- $\mathcal{B}$ sends to $\mathcal{A}$ the public key (g_1, g_2, u, g_2^y, z)
- $\mathcal{A}$ starts to work. It requests a signature for w_i:
  - $\mathcal{B}$ responds with σ_i and r_i where $m_i + r_i \cdot y = w_i$ (so $\sigma_i = g_1^{1/(w_i+x)} = g_1^{1/(m_i+x+r_i \cdot y)}$)
- finally $\mathcal{A}$ forges a signature $\sigma_* = g_1^{1/(m_*+x+r_* \cdot y)}$ which is a signature for the reduced scheme for message $m_* + r_* \cdot y$

  existential forgery suceeded!

recall Schnorr signature

network protocols, Bitcoin, ...

Setup:

g- generator of a subgroup of prime order q, computations $\bmod p$ where p prime

- private key: $x < q$ chosen at random
- public key: $X = g^x \bmod p$

Signature creation:

i. $k < q$ chosen at random,

ii. $r := g^k \bmod p$

iii. $e := \mathrm{Hash}(M, r)$

iv. $s := k - e \cdot x \bmod q$

v. output (s, e)

Verification:

signature valid iff $e = \mathrm{Hash}(M, g^s \cdot X^e)$

No security proof in the standard model

consequence: weakness of Hash $\Rightarrow$ disaster

example: assume that for $e = \mathrm{Hash}(M, r)$ one can find M', δ such that $e \cdot \delta = \mathrm{Hash}(M', r^{\delta})$

then $s \cdot \delta = k \cdot \delta - (e \cdot \delta) \cdot x \bmod q$

$$s \cdot \delta = (k \cdot \delta) - \mathrm{Hash}(M', g^{k \cdot \delta})$$

and one can present a signature for M' based on signature for M without using x:

i. recompute $r = g^k$ from (s, e)

ii. find M', δ

iii. output $s' = s \cdot \delta \bmod q$, $e' = \mathrm{Hash}(M', r^{\delta})$

Schnorr Signatures in Random Oracle Model

assume that algorithm A can forge signatures for ROM for Hash

scenario:

- device running A
- oracle for Hash running according to ROM

derivation of secret key:

i. run the algorithm forging a signature, (s, e) obtained for M

   – at some moment A asked oracle for $\mathrm{Hash}(M, r)$ where $r = g^s \cdot X^e$, otherwise the hash value would be not fixed and during verification set to something probably $\neq e$

   – so at this moment r is already fixed!

ii. rewind the forging algorithm to the moment of asking the oracle: restore the same state of all registers, continue

- the answer for $\mathrm{Hash}(M, r)$ changed to e'
- the signature afterwards is computed in deterministic way: $s' = \log(r) - e' \cdot x \bmod q$

  (the previous was: $s = \log(r) - e \cdot x \bmod q$

iii. solve:

$$s - s' = (\log(r) - \log(r)) - (e - e') \cdot x \bmod q$$

$$x = (s - s')/(e' - e) \bmod x$$

One issue left:

– maybe the forger needs some valid signatures to create the next one

– so to forge a signature we need first to forge signatures in order to reduce to Discrete Logarithm Problem?

Solution: simulation of signatures in ROM

i. choose s, e at random

ii. compute $r := g^s \cdot X^e$

iii. insert e in the hashtable for argument (M, r)

(if already there is an entry for (M, r) then backtrack)

Signing Anonymous Transactions

idea:

- transactions records publicly available in a distributed ledger (DLT) $\Rightarrow$ undeniability, no backdating, possibility to detect double spending , against Money Laundering
- however, we must not create a public Big Brother

core mechanism for digital currencies:

cash hides money flow, this should be the key property of digital money as well

examples below will be taken from Monero

User keys and hidden recipient

user keys (EC notation):

- private keys a, b
- public keys: $A = a \cdot G$, $B = b \cdot G$
- sometimes (a, B) revealed (tracking key) – if the transactions have to be deanonymized

Creating transaction with a hidden recipient: (Alice sends to Bob)

- Alice fetches the public key (A, B)
- Alice chooses r at random, $R := r \cdot G$
- Alice generates one-time public key $P := \mathrm{Hash}(r \cdot A) \cdot G + B$
- Alice uses P as a one-time destination key for the transaction containing metadata R

Receiving a transaction by Bob

- Bob tries each transaction posted:
  - → compute $P' := \mathrm{Hash}(a \cdot R) \cdot G + B$
  - → if this is the right transaction, then $P = P'$ and Bob knows it is for him
- Bob calculates the one-time private key:

$$x = \mathrm{Hash}(a \cdot R) + b$$

- Bob can spend the money obtained in the transaction by signing with x

Remarks:

1: Receiving a transaction possible with (a, B), while (a, B) does not enable to compute x

2: Still only a partial anonymity: using x and the public key P would indicate who has got transaction with P from Alice

One time ring signatures

idea:

- instead of signing with x and showing P, a ring signature created:
  - a set of public keys $P_1, P_2, ..., P_m$ from transactions chosen at random (transaction value must be the same)
  - x used for signing
- any two ring signature of this kind created with x will be linked immediately

Goals achieved:

- double spending exposed
- m-anonymity concerning where the e-coin comes from

Creating one-time ring signature

for key pair (x, P)

1. compute image key

$$I := x \cdot \mathrm{Hash}(P)$$

2. choose a ring of keys $P(P_{0,}...., P_n)$ where $P_s = P$ for some s
3. choose $q_0, ..., q_n$ at random
4. choose $w_0, ..., w_n$ at random, except for w_s
5. calculate for $i \neq s$

$$L_i := q_i \cdot G + w_i \cdot P_i$$

6. calculate $L_s := q_s \cdot G$

7. calculate for $i \neq s$

$$R_i := q_i \cdot \mathrm{Hash}(P_i) + w_i \cdot I$$

8. calculate $R_s := q_s \cdot \mathrm{Hash}(P_s)$
9. calculate the non-interactive challenge:

$$c := \mathrm{Hash}(\mathrm{message}, L_0, ..., L_n, R_0, ..., R_n)$$

10. calculate individual components:
    - for $i \neq s$: $c_i = w_i$, and $r_i = q_i$
    - $c_s := c - \sum_{i \neq s} c_i$
    - $r_s := q_s - c_s \cdot x$
11. output signature $(I, c_0, ..., c_n, r_0, ..., r_n)$

Verification

L_i recomputed as $L'_i := r_i \cdot G + c_i \cdot P_i$

R_i recomputed as $R'_i := r_i \cdot \mathrm{Hash}(P_i) + c_i \cdot I$

test:

$$\sum c_i = \mathrm{Hash}(\mathrm{message}, L'_0, ..., L'_n, R'_1, ..., R'_n)$$

Linking:

via the same I

Concept used:

to close the ring somewhere a schnorr signature must be created that applies to two generators simultaneously:

- P_s (which is hidden)
- I (which is explicit)

Many extensions possible (e.g. a transaction signed with multiple keys)

Shortcomings of anonymous transactions with ring signatures

crypto is the strongest component, the problem is elsewhere:

1. the size of the ring is relatively small (e.g. 5, 10)
2. if Bob floods the system with own transactions (e.g. 50% of volume), then the effective size of each ring is substantially reduced from the point of view of B
3. how to choose the keys for the ring?
   - → only the recent ones? then making atranaction with an old key reveals this key
   - → all keys with the same probability? Then fast transactions exposed
   - → ... many issues for traffic analysis
4. **Conclusion:** do not trust anonymous transactions blindly

Deterministic versions - protection against weak PRNG implementations

predictable k in case of Schnorr: $s = k - e \cdot x \bmod q$ enables to solve for x

Idea: remove randomness from signatures

example 1: EdDSA

like EC Schnorr signature on Eduards curve where

$$k := \mathrm{Hash}(x_1, M)$$

where x_1 is a part of the secret key and M is the message to be signed

other part for computing $s = k - x_0 \cdot e \bmod e$

BLS signature (Boneh–Lynn–Shacham)

for bilinear groups: pairing $e\colon G \times G \to G_T$, secret key sk, public key $\mathrm{pk} = g^x$

signature creation:

$$\mathrm{signature}(M) := \mathrm{Hash}(M)^x$$

verification: of s for M

$$e(s, g) = e(\mathrm{Hash}(M), \mathrm{pk})?$$

Deterministic EC DSA (popular for cryptocurrency wallets)

Setup:

- elliptic curve E, point $G \in E$ of prime order q (at least 160 bits)
- private key - random number $x < q$
- public key $U = x \cdot G$

EC DSA signature creation

i. $h := \mathrm{Hash}(M) \bmod q$

ii. k generated at random, calculate $k \cdot G$,

iii. $r := x\,\mathrm{coordinate}(k \cdot G) \bmod q$

iv. $s := (h + x \cdot r)/k \bmod q$

v. output (r, s)

deterministic version: the same but k is an output of HMAC (see below)

HMAC computation

i. $h_1 := \mathrm{Hash}(M)$

ii. $V := 0x010x01 \ldots 0x01$, $K := 0x000x00 \ldots 0x00$

iii. $K := \mathrm{HMAC}_K(V, 0x00, x, h_1)$, $V := \mathrm{HMAC}_K(V)$

iv. $K := \mathrm{HMAC}_K(V, 0x01, x, h_1)$,, $V := \mathrm{HMAC}_K(V)$

v. initialize empty string T

- run a loop until enough bits generated (the same length as q):
- $V := \mathrm{HMAC}_K(V)$, $T := T \| V$

vi. if $T < q$ then $k := T$, else restart with $K := \mathrm{HMAC}_K(V, 0x00)$, $V := \mathrm{HMAC}_K(V)$

CRYSTALS Dilithium

"post-quantum" algorithm, finalist of NIST competition,

based on lattice crypto

Problem (easy) for linear algebra:

$\mathbf{A} \cdot \mathbf{y} = \mathbf{z} \bmod p$, given matrix $\mathbf{A}$ and vector $\mathbf{z}$ find vector $\mathbf{y}$

Problem - hard:

$\mathbf{A} \cdot \mathbf{y} + \mathbf{e} = \mathbf{z} \bmod p$, given matrix $\mathbf{A}$ and vector $\mathbf{z}$ find vector $\mathbf{y}$ and $\mathbf{e}$ that are small

all solutions $(\mathbf{y}, \mathbf{e})$ form a "lattice"

Algebra used:

- prime $q = 2^{23} - 2^{13} + 1$,
- $n = 256$, ring $R = \mathbb{Z}_q[X]/(X^n + 1)$

Key generation

Gen

01 $\mathbf{A} \leftarrow R_q^{k \times \ell}$

02 $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow S_\eta^\ell \times S_\eta^k$

03 $\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$

04 **return** $(pk = (\mathbf{A}, \mathbf{t}), sk = (\mathbf{A}, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2))$

– vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ are vectors from R with coefficients $< \eta$ (small)

Signature generation

$\underline{\mathsf{Sign}(sk, M)}$

05 $\mathbf{z} := \perp$
06 **while** $\mathbf{z} = \perp$ do
07 $\quad \mathbf{y} \leftarrow S^{\ell}_{\gamma_1 - 1}$
08 $\quad \mathbf{w}_1 := \mathsf{HighBits}(\mathbf{A}\mathbf{y}, 2\gamma_2)$
09 $\quad c \in B_\tau := \mathrm{H}(M \parallel \mathbf{w}_1)$
10 $\quad \mathbf{z} := \mathbf{y} + c\mathbf{s}_1$
11 $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathsf{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2, 2\gamma_2)\|_\infty \geq \gamma_2 - \beta$, then $\mathbf{z} := \perp$
12 **return** $\sigma = (\mathbf{z}, c)$

- security parameters γ_1, γ_2 , $\beta =$ maximum possible coefficient of $c \cdot \mathbf{s}_i$ $(< \eta \cdot \tau)$
- $\mathbf{y}$ - consists of small elements
- $\mathbf{w}_1$ every number in the result truncated to high bits
- c contains τ elements ± 1, otherwise zeroes
- condition in 11: if any coefficient exceeds the threshold then backtrack , parameters chosen so that a fair probability to suceed

Signature verification

$\mathsf{Verify}(pk, M, \sigma = (\mathbf{z}, c))$

13 $\mathbf{w}_1' := \mathsf{HighBits}(\mathbf{Az} - c\mathbf{t}, 2\gamma_2)$

14 **if return** $[\![\|\mathbf{z}\|_\infty < \gamma_1 - \beta]\!]$ **and** $[\![c = \mathrm{H}(M \parallel \mathbf{w}_1')]\!]$

Why $\mathbf{w}_1' = \mathbf{w}_1$?

$\mathrm{HighBits}(\boldsymbol{A} \cdot \boldsymbol{z} - c \cdot \mathbf{t}) = \mathrm{HighBits}(\boldsymbol{A} \cdot \mathbf{y})$?

$\boldsymbol{A} \cdot \boldsymbol{z} - c \cdot \mathbf{t} = \boldsymbol{A} \cdot (\mathbf{y} + c \cdot \mathbf{s}_1) - c \cdot \mathbf{t} = \boldsymbol{A} \cdot \mathbf{y} + \boldsymbol{A} \cdot c \cdot \mathbf{s}_1 - c \cdot \mathbf{t} = \boldsymbol{A} \cdot \mathbf{y} - c \cdot \mathbf{s}_2$

$c \cdot \mathbf{s}_2$ has all coefficients $< \beta$, while $\mathrm{LowBits}(\boldsymbol{A} \cdot \mathbf{y} - c \cdot \mathbf{s}_2)$ are $< \gamma_2 - \beta$

Practical Challenge - size of the (public) keys and signatures

Further optimization to reduce the public key:

- $\mathbf{A}$ generated from a small seed on-the-fly
- not the whole $\mathbf{t}$, only $\mathbf{A} \cdot \mathbf{s}_1 + \mathbf{s}_2 = b \cdot \mathbf{t}_1 + \mathbf{t}_0$
- signer has to provide a help vector of carry bits to mimic computation of HighBits

Security Issues

Quantum Random Oracle? – computations taking into account many hash computations at once

LWE Problem - Learning with Errors it is infeasible to distinguish between

- $(\mathbf{A}, \mathbf{u})$ where $\mathbf{u}$ chosen uniformly at random
- $(\mathbf{A}, \mathbf{t})$ where $\mathbf{t} = \mathbf{A} \cdot \mathbf{s}_1 + \mathbf{s}_2$ for small $\mathbf{s}_1, \mathbf{s}_2$

SelfTargetMSIS

- given M,$\mathbf{A}, \mathbf{t}$
- find c and a small norm vector $\mathbf{y}$ such that $c = \mathrm{Hash}(M, [\mathbf{A}|\mathbf{I}|] \cdot \mathbf{y} + c \cdot \mathbf{t})$

related problem MSIS:

find $\mathbf{y}$ with small coefficients such that $[\mathbf{A}|\mathbf{I}|] \cdot \mathbf{y} = \mathbf{0}$

ongoing research about assumptions ...

PSEUDONYMOUS SIGNATURES

Application areas:

- while having the pseudonyms, how to authenticate digital data? Digital signatures would solve the problem
- implementing GDPR rights in practice:

  a data subject can authenticate the request (e.g. for data rectification) in a database with pseudonyms by sending a request with a signature corresponding to the pseudonym

BSI Pseudonymous Signature:

- **keys:**
  - domain parameters D_M and a pair of global keys $(\mathrm{PK}_M, \mathrm{SK}_M)$
  - public key $\mathrm{PK}_{\mathrm{ICC}}$ for a group of eIDAS tokens, the private key $\mathrm{SK}_{\mathrm{ICC}}$ known to the issuer of eIDAS tokens
  - assigning the private keys for a user:

    the issuer chooses $\mathrm{SK}_{\mathrm{ICC},2}$ at random, then computes $\mathrm{SK}_{\mathrm{ICC},1}$ such that

    $$\mathrm{SK}_{\mathrm{ICC}} = x_1 + \mathrm{SK}_M \cdot x_2$$
  - a sector (domain) holds private key $\mathrm{SK}_{\mathrm{sector}}$ and public key $\mathrm{PK}_{\mathrm{sector}}$.
  - a sector has revocation private key $\mathrm{SK}_{\mathrm{revocation}}$ and public key $\mathrm{PK}_{\mathrm{revocation}}$
  - sector specific identifiers I_1^{sector} and I_2^{sector} for the user:

    $$I_1^{\mathrm{sector}} = (\mathrm{PK}_{\mathrm{sector}})^{x_1}$$
    $$I_2^{\mathrm{sector}} = (\mathrm{PK}_{\mathrm{sector}})^{x_2}$$

- **signing:** with keys x_1, x_2 and I_1^{sector} and I_2^{sector} for $\text{PK}_{\text{sector}}$ and message m

  i. choose K_1, K_2 at random

  ii. compute

  - $Q_1 = g^{K_1} \cdot (\text{PK}_M)^{K_2}$
  - $A_1 = (\text{PK}_{\text{sector}})^{K_1}$
  - $A_2 = (\text{PK}_{\text{sector}})^{K_2}$

  iii. $c = \text{Hash}(Q_1, I_1^{\text{sector}}, A_{1,2}^{\text{sector}}, A_2, \text{PK}_{\text{sector}}, m)$

  (variant parameters omitted here)

  iv. compute

  - $s_1 = K_1 - c \cdot x_1$
  - $s_2 = K_2 - c \cdot x_2$

  v. output (c, s_1, s_2)

- **verification:**

  compute

  - $Q_1' = (\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{s_1} \cdot (\mathrm{PK}_M)^{s_2}$
  - $A_1' = (I_1^{\mathrm{sector}})^c \cdot (\mathrm{PK}_{\mathrm{sector}})^{s_1}$
  - $A_2' = (I_2^{\mathrm{sector}})^c \cdot (\mathrm{PK}_{\mathrm{sector}})^{s_2}$
  - recompute c and check against the c from the signature

- why it works?

$$(\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{s_1} \cdot (\mathrm{PK}_M)^{s_2} = (\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{K_1} \cdot (\mathrm{PK}_M)^{K_2} \cdot g^{-c \cdot x_1} \cdot (\mathrm{PK}_M)^{c \cdot x_2}$$

$$= (\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{K_1} \cdot (\mathrm{PK}_M)^{K_2} \cdot g^{-c \cdot x_1} \cdot (g)^{-c \cdot \mathrm{SK}_M \cdot x_2}$$

$$= (\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{K_1} \cdot (\mathrm{PK}_M)^{K_2} \cdot g^{-c \cdot \mathrm{SK}_{\mathrm{ICC}}} = g^{K_1} \cdot (\mathrm{PK}_M)^{K_2} = Q_1$$

- there is a version without A_1, A_2 and the pseudonyms $I_1^{\mathrm{sector}}, I_2^{\mathrm{sector}}$

Problems:

- **the issuing authority knows the private keys**

  but: there is a way to solve it when the user gets two pairs of keys on the device and takes their linear combination)

- **breaking into just 2 devices reveals the system keys**