

Lecture Notes

e€

(electronic money/cash)

Michael Nüsken

b-it

(Bonn-Aachen International Center
for Information Technology)

IPEC winter 2008

e€ Expectations

Have a good
time

Something about
Crypto

REPETITION
FOR ELECT

Interesting
Facts about
electronic payment

4 Credit-Points
and Crypto

1. Good grade
2. Something totally (very) new
3. 4 more Credits

- good grade
- crypto details
- programming & practical stuff

FUN + SEE
WHY e-CASH
Works

- to learn
from you

learn in a nice
environment

HAVE A NICE
WEEK

learn everything
about electronic
cash

Last 40p is A
and much fun!

A-Schein

A-Schein
+ some Exp. in crypto

e €

PE
31.3.08
①

What is money?

certificate

face of ruler
or banner

trade
business exchange

coin =
metal value

agreement

laws future

symbol

punishment trust

gold equivalence

contract

value
worth

promise

Scottish money reads sth like "Show this banknote
(British?) to ... and you'll get ..."

Forms of money

e€
31.3.0P
(2)

- Coin
 - Bills, banknotes
 - Bank account
 - Stocks, ...
- } "hardware" money
cash
- } "software" money
- not 0 4 P 2

On the internet? e-business?

→ usually account-based transaction
(by withdrawal, bank transfer)
Also pay pal is a bank.

Distinctions

- online / offline
(takes a few days...)
- anonymous / non-a.

? anonymous + electronic?

Do we want anonymity?

- Yes, to prevent the account holder (bank) to gather too much information about its customers.

- No, because of ~~public~~ ^{social} security.

- No, because of possibility of loss.

Schneier about privacy vs. security.

"Freedom is fragile.
Handle it with care."

Physical cash identifiers

- Watermark
- Special paper
- UV markers
- silver stripe
- special colors
- serial number
- image, fingerprint
- bank representative signature
- relief print
- front and back print fit together
- bank name, currency name

e€
31.3.08

(3)

MitHagsschlaf

€
ToDo

We need
to learn

It might
be good to
know more
about...

Probability Theory
(exponential distribution)

RSA

ENCRYPTION

El Gamal

encryption?

RSA

SIGNATURE

unconditionally-
vs.
computationally secure

homomorphic
encryption

Hamming-Distance

Groups

Rings/Fields

Blind signatures
(blinded)

Schnorr signature
scheme

$\phi(n)$

(n = RSA modulus)

"cut-and-choose"
methodology

minimum disclosure,
zero-knowledge

decision "Diffie-
Hellman" assumption
?

zero-knowledge
proofs?

Tamper-resistant
smart-

Certificates

Schnorr proof
of knowledge?

Block cipher
(DES/AES)

One-way-function
collision-freeness
"similarity to a random oracle"

Hash-function

Major obstacle to make it electronically
Bit strings can be copied easily!

Money NEVER!

CE
31.3.08
(4)

Plan for the week

PKC	PKC + EX + more basic prog.	Great Congress	IMPLEMENTATION + Background	
PKC				
MAD				

Public key cryptography

€
31.03.08
(5)

- Ring of Integers modulo some number N :

$$\mathbb{Z}_N, +, -, \cdot, \text{ sometimes } ?^{-1}$$

$\rightarrow \mathbb{Z}_N^*$ unit group (invertible elements)

$$= \{ \underset{\substack{\uparrow \\ \in \mathbb{Z}}}{x \bmod N} \mid \gcd(x, N) = 1 \}$$

(to the contrary: $x \bmod N \in \mathbb{Z}$.)

For inversion and to determine gcds we use the EEA (extended Euclidean Alg.)

$$\boxed{r_i = s_i a + t_i b}$$

r_i	q_i	s_i	t_i
$a = 12$		1	0
$b = 9$		0	1
$\boxed{3}$	3	1	-1
0		-3	4

gcd!

Thus we get

$$\gcd(a, b) = s \cdot a + t \cdot b$$

Reading this modulo a :

$$\gcd(a, b) = t \cdot b \equiv 1 \Rightarrow \text{inverse of } b$$

Cross checking

$$-9/3 \quad 12/3$$

~~Groups~~

Then \mathbb{Z}_N is a field, i.e. $\mathbb{Z}_N^* = \mathbb{Z}_N \setminus \{0\}$
 [iff N is prime

• Groups (commutative)

Proper: set with one operation, well def'd

c €
31.03.0P
(6)

A : associative

N : neutral

I : inverses exist

C : commutative

Examples: (\mathbb{R}^x, \cdot) , $(\mathbb{R}, +)$

$(\mathbb{Z}, +)$, $(\mathbb{Z}^x, \cdot) = (\{\pm 1\}, \cdot)$

$(\mathbb{Z}_N, +)$, (\mathbb{Z}_N^x, \cdot)

mostly be prime
or a product of
two primes

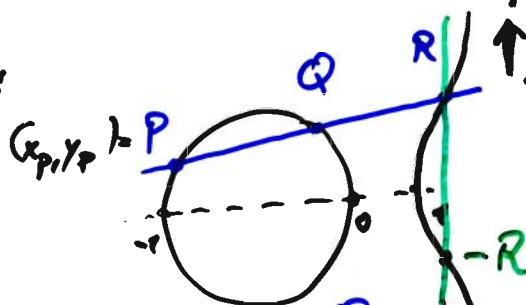
EC groups

set: set of solutions of an
equation $y^2 = x^3 + ax + b$
with $a, b \in \mathbb{F}$
where \mathbb{F} is some field,

(finite)

eg. $\mathbb{F} = \mathbb{Z}_p$ with p prime

operation:



real EC
 $y^2 = x^3 + x$

define: $P + Q + R = O$

ie. we want $P + Q = -R$

Exponentiation in groups:

Say you are ^{given} an element g in a group G .

Then for any $\alpha \in \mathbb{Z}$ you can compute

$$g^\alpha \in G.$$

(To do that fast use Square and Multiply,
ie. $O(n)$ group operations
 \uparrow
 # bits of α .)

If G is finite we have

Then (Lagrange) If G is a finite group

$$\left[\begin{array}{l} \text{then } g^{\#G} = 1_G \\ \text{for every } g \in G. \end{array} \right. \quad \triangle$$

In case $G = \mathbb{Z}_N^\times$ unit group of integers mod N
we get

Then (Euler) If x is coprime to N , ie $x \in \mathbb{Z}_N^\times$

$$\text{then } x^{\varphi(N)} = 1 \text{ in } \mathbb{Z}_N,$$

where φ is the EULER totient function,

$$\varphi(N) := \# \mathbb{Z}_N^\times.$$

$$2^{1234567} \bmod 11$$

Lemma p prime: $\varphi(p) = p-1$,

p, q prime, $p \neq q$: $\varphi(pq) = (p-1) \cdot (q-1)$.

$$2^{7 \bmod 11}$$

Then (Little Fermat) If p is prime, $x \in \mathbb{Z}_p, x \neq 0$ then $x^{p-1} = 1 \in \mathbb{Z}_p$.

So

$$(\mathbb{Z} \# G)^+ \xrightarrow{\text{Lagrange}} (G, \cdot)$$

e €
37.03.08
8

$$\text{exp}_g: \begin{matrix} \alpha \\ \alpha + \beta \end{matrix} \xrightarrow{\quad} \begin{matrix} g^\alpha \\ g^{\alpha + \beta} = g^\alpha \cdot g^\beta \end{matrix}$$

is well def'd by thm (Lagrange) and a homomorphism

↑ If operation is addition this looks slightly different:

$$\mathbb{Z} \# G \longrightarrow E$$

$$\alpha \longmapsto \alpha \cdot p$$

is well def'd by thm (Lagrange). In particular:

$$\#G \cdot p = 0_E$$

If this map is surjective then we say:

G is generated by g ,

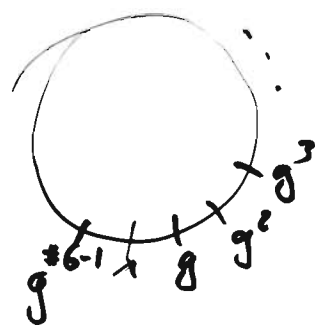
G is cyclic.

Then exp_g is an isomorphism.

Only: we know that computing $\text{exp}_g(x)$ is cheap but

Discrete
Logarithm
Problem

computing α from $\text{exp}_g(\alpha) = x$
may be extremely difficult.



Assume that G is a cyclic finite group.

c €
31.03.08
(9)

Then $\exp_g : \mathbb{Z}_{\#G} \xrightarrow{\alpha} G$
 $\xrightarrow{\quad} g^x$

is an isomorphism and cheap.

For many groups finding α such $g^\alpha = x$
is difficult.

Necessary : $\#G$ has a large prime factor.

Interesting: for $G = \mathbb{Z}_p^*$, p prime
the best known ~~attacks~~ algorithms
to solve the DLP
with $p \sim 2^{1024}$
are approximately as ~~diff~~ fast
as the best known algorithms
(generic) for a group G with $\#G \sim 2^{160}$.
(Thus we prefer elliptic curves in many
situations..)

Some examples

$e \in 31.03.08$
(10)

ring of integers:

$$\mathbb{Z}_{13} = (\{0, 1, 2, 3, \dots, 12\}, +, \cdot)$$

$$\mathbb{Z}_{13}^{\times} = (\{1, 2, 3, \dots, 12\}, \cdot) \cong (\mathbb{Z}_{12}, +)$$

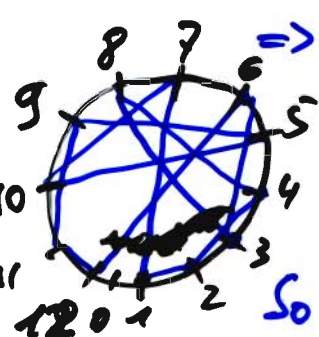
$$\mathbb{Z}_6 = (\{0, 1, 2, 3, -2, -1\}, +, \cdot)$$

$$\mathbb{Z}_6^{\times} = (\{1, -1\}, \cdot) \cong (\mathbb{Z}_2, +)$$

$$(-1)^{\alpha} \xleftarrow{\exp_{-1}} \alpha$$

$$\left. \begin{array}{l} 2^6 = -1 \neq 1 \\ 2^4 = 3 \neq 1 \\ 2^{12} = 1 \text{ (Lagrange!)} \end{array} \right\} \Rightarrow 2 \text{ generates } \mathbb{Z}_{13}^{\times}$$

$$\begin{array}{l} 2^{\alpha} = 1 \\ 0 < \alpha < 12 \end{array} \Rightarrow \begin{array}{l} 2^{s \cdot \alpha + t \cdot 12} = 1 \\ 2^{\gcd(k, 12)} \end{array} : \exists \alpha \mid 12.$$

 \Rightarrow a multiple of α is just a prime away of 12, i.e. $\exists \alpha = \frac{12}{2}$ or $\alpha = \frac{12}{3}$.

So since $2^4 \neq 1$ and $2^6 \neq 1$ no such α exists.

\mathbb{Z}_{12}	0	1	2	3	4	5	6	-5	-4	-3	-2	-1	0
\mathbb{Z}_{13}^{\times}	1	2	4	8	3	6	12	11	5	10	9	7	1

RSA (Rivest, Shamir & Adleman '78) e ∈ 31.03.08
(11)
already known to CSES which is part of British Secret Service around 1972

Choose two (different) primes p, q .

Compute $N := p \cdot q$,

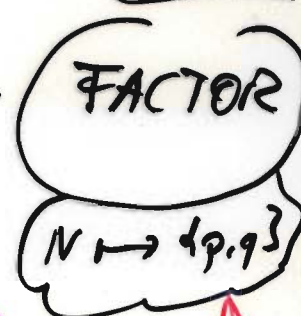
$$L := (p-1) \cdot (q-1).$$

Further, choose e, d such that $e \cdot d \equiv 1 \pmod{L}$ by EEA!

$$e \cdot d = 1 \text{ in } \mathbb{Z}_L.$$

Save the private key (N, d) and publish the public key (N, e) .

Discard everything else.



Encrypt a message $x \in \mathbb{Z}_N$:

Compute $y = x^e$ in \mathbb{Z}_N .

Note: you only need the public key.



Decrypt a ciphertext $y \in \mathbb{Z}_N$:

Compute $z = y^d$ in \mathbb{Z}_N .

Note: you need the private key here.

Observe:

CORRECTNESS $z = y^d = x^{ed} = x^{1+kL} = x$

EFFICIENCY? ✓



... at least for $x \in \mathbb{Z}_N^*$...

El Gamal encryption

e€
31.03.08
(12)

Fix a group G with generator g
so that the DLP is difficult.

For example: choose q a 160-bit prime,
 p a 1024 bit prime
with $p \equiv 1 \pmod{q}$
i.e. $p = 1 + a \cdot q$
for some a .

and choose $h \in \mathbb{Z}_p^\times$ at random
and compute $g = h^{\frac{p-1}{q}}$,
if $g \neq 1$ that's our generator g .
Otherwise try another h .

We let $G = \langle g \rangle = \{1, g, g^2, \dots, g^{q-1}\}$.

Choose $\alpha \in \mathbb{Z}_q$ and let $a = g^\alpha$.

Encrypt $x \in G$ (or $x \in \mathbb{Z}_p^\times$).

Choose $\tau \in_R \mathbb{Z}_q$.

Compute $t = g^\tau$, $y = \frac{a^\tau}{g^{\tau\alpha}} \cdot x$

CORRECT

Decrypt $(t, y) \in G \times G$.

Compute $y / t^\alpha = y / g^{\tau\alpha} = a^\tau x / a^\tau = x$.

EFFICIENCY? ✓

SECURITY?

Tomorrow

Add-ons

e€
1.04.08

①

Primality testing (simple)

Fermat test

Input: N a number.

Output: • Yes, it may be prime,

• No, it is not prime.

0. If $\gcd(N, 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11) \neq N, 1$ then Return NO!

1. Choose x with $0 < x < N$ at random.

2. Compute $u := x^{N-1} \bmod N$. $r = (x \bmod N)^{N-1}$.

3. If $u \neq 1$ then Return NO.

else Return YES.

Strong Fermat test

Input: N a number

Output: YES, NO.

expose(isprime)

0. Sort out small prime factors.

1. Choose x with $0 < x < N$ uniformly at random.

2. Write $N-1 = 2^r \cdot s$ with s odd.

3. Compute $(x \bmod p)^s, (x \bmod p)^{2s}, \dots, (x \bmod p)^{2^{r-1}s}$.

4. If none of these = 1 then NO!

5. If $(x \bmod p)^s = 1$ then Yes

6. If before the first 1 we see a -1 then NO!

7. Yes.

Then: If N is prime then also returns Yes. Otherwise it returns NO with probability $\geq \frac{1}{2}$ (3/4)

Prime Number Theorem

es
01.04.08

(2)

Denote $\pi(x) := \#\{p \text{ prime} \mid p \leq x\}$

Then

$$\pi(x) \sim \frac{x}{\ln x}$$

or

$$\frac{\pi(x)}{x} \sim \frac{1}{\ln x} = \frac{1/\ln 2}{\log_2 x} \cdot \frac{\log_2 e}{\log_2 x}$$

more precisely:

$$\pi(x) = \frac{x}{\ln x} \left(1 + k(x) \cdot \frac{1}{\ln x} \right)$$

$$\text{with } \frac{1}{2} < k(x) < \frac{3}{2}$$

Consequently,

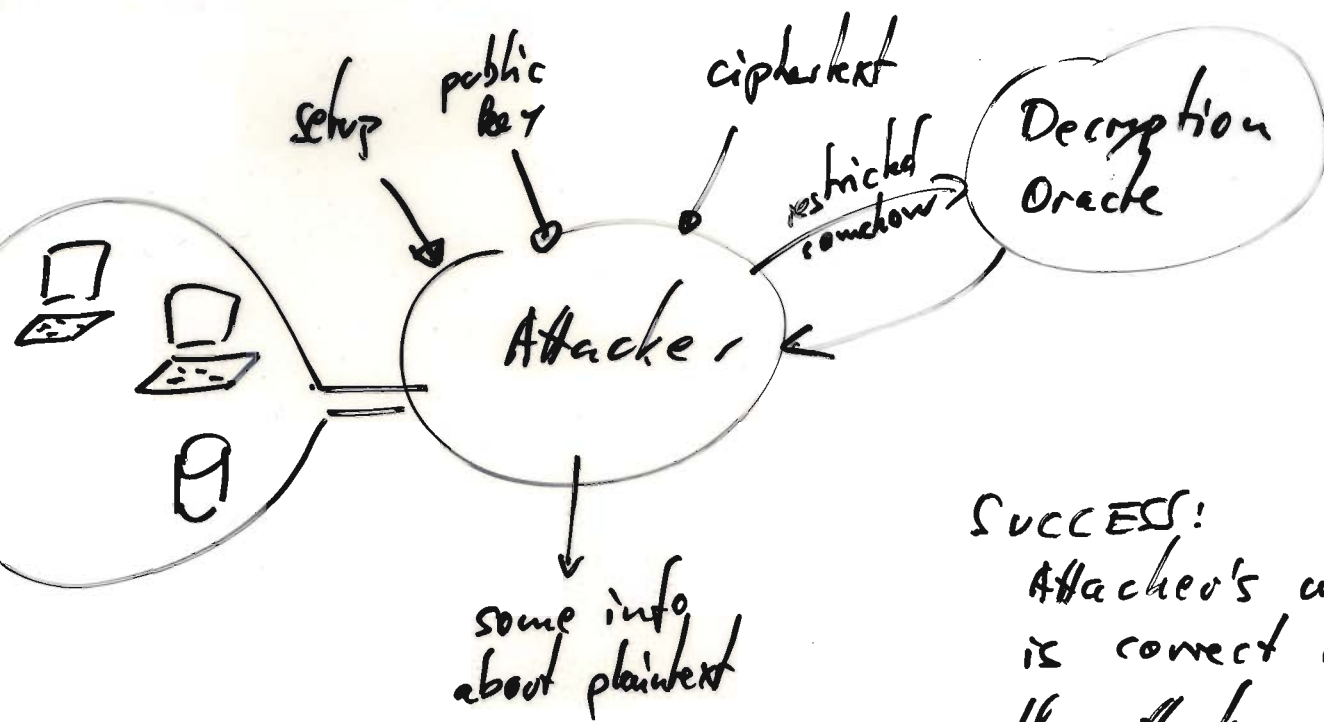
searching for an n -bit prime
we expect $n \cdot \ln 2$ random numbers to test
~~will need~~
and each test (Strong Fermat test)
costs at most $O(n^3)$ operations.

So finding a prime costs

expected $O(n^4)$ operations.

Security notion

€
1.01.08
(3)

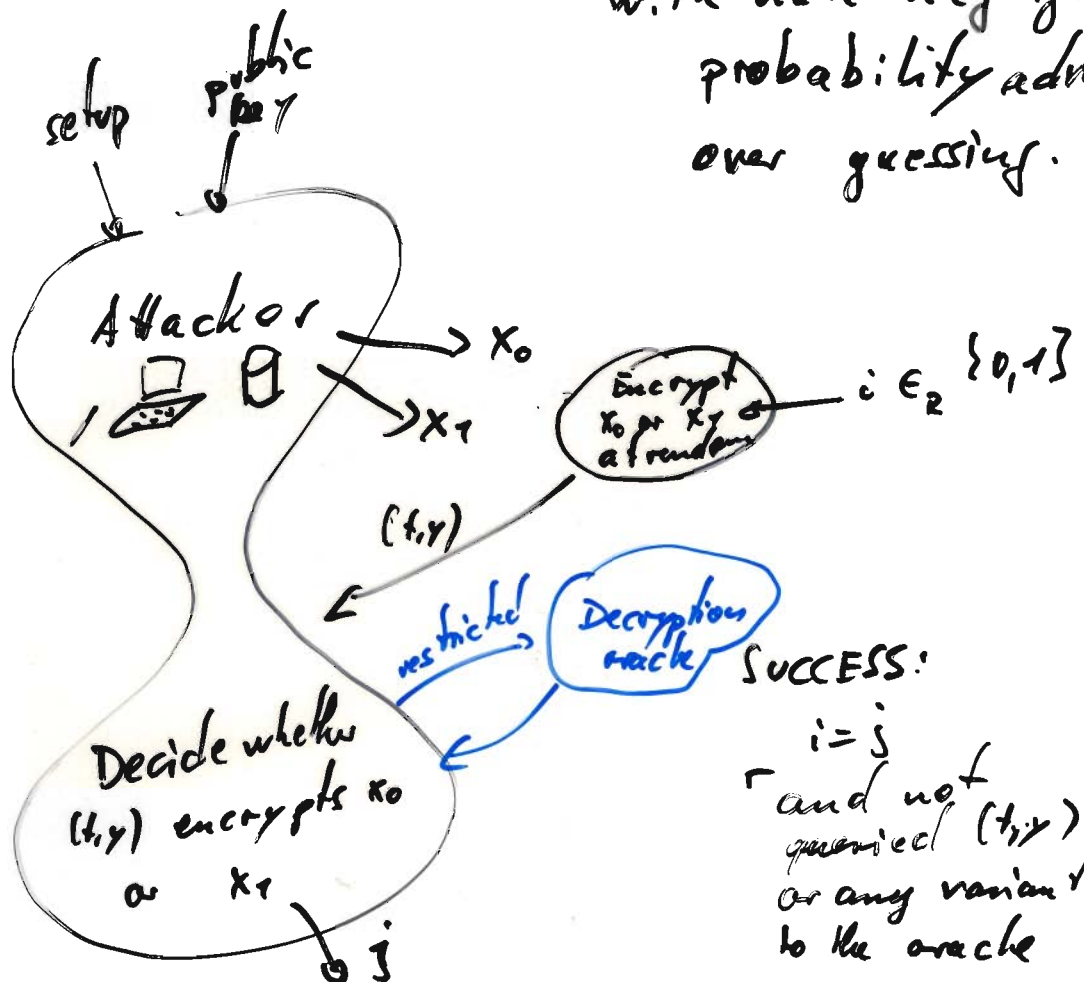


SUCCESS!

Attacker's answer is correct and the attacker never asked the Decryption oracle on the given ciphertext.

with non-negligible probability advantage over guessing.

Even stronger!



SUCCESS!

$i = j$
and not queried (t, y) or any variant of it to the oracle

Possibly difficult problem!

1.04.08
(4)

DLP : $g^x \mapsto x$
(Discrete Log Problem)

DHP : $(g, g^P, g^Q) \mapsto g^{PQ}$
(Diffie Hellman Problem)

DDHP : $(g, g^P, g^Q, g^S) \mapsto$ $S = PQ?$
(decisional DHP)
 $g^S = g^{PQ}?$

Solution to DHP \Rightarrow Solution to DDHP
 \uparrow

Solution to DLP

Assume we have an attacker who can always distinguish an encryption of x_0 from an encryption of x_1 where itself chose x_0 and x_1 .

ElGamal enc
 $a = g^x$
 $t = g^z$
 $y = a^z \cdot x$

Reduction

Input: (g, b, c, d)

Output: Yes/No and should answer the question:
 $d = g^{PQ}$ when $b = g^P, c = g^Q$.

1. Define setup: use the group in question; $G = \langle g \rangle$.
2. Let $a = b$, $a = b g^S$
3. Ask the attacker for x_0 and x_1 .
4. Choose $i \in_R \{0, 1\}$. $t = c g^i$ $y = d g^i x_i$
5. Let $t = c$ and $y = d \cdot x_i$.
6. Ask the attacker which plaintext is encrypted by (t, y) . Say it says $j \in \{0, 1\}$.
7. Return $(i = j)$

z , properly
chose
makes other
input boxes
distributed.

Case The correct answer is Yes.

e €
1.04.08
(5)

$$\text{prob}(\text{We say yes})$$

$$= \text{prob}(i = \hat{j})$$

$$= \text{prob}(\text{Attacker succeeds}) \approx$$

$$\begin{cases} = 1 & \text{assuming the attacker is perfect.} \end{cases}$$

$$\begin{cases} > \frac{1}{2} + \frac{1}{n^c} & \text{assuming the attacker has a non-negligible success probability} \end{cases}$$

Case d is random. (that's almost the other case)

The attacker assumes that $y = g^{\beta x} \cdot x'$
 $= d \cdot x_i$

$$\hookrightarrow x' = g^{-\beta x} d \cdot x_i \leftarrow \text{random.}$$

\uparrow random \nearrow

So the attacker sees sth random encrypted.

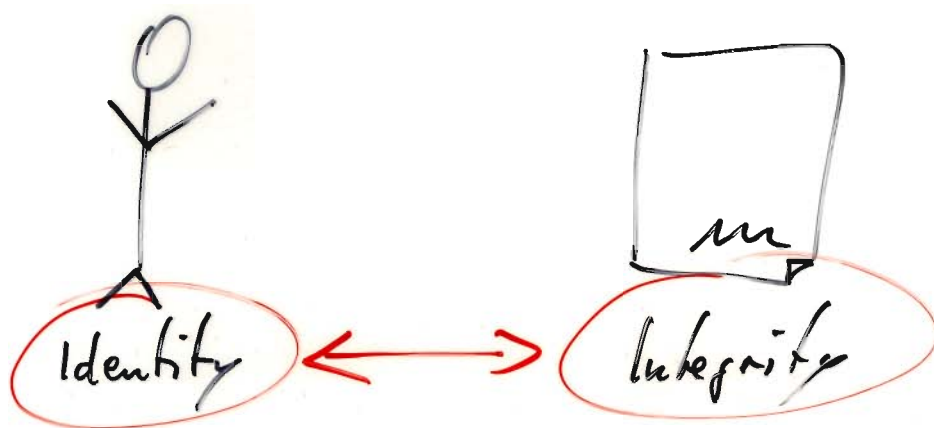
So its answer is like a guess.

$$\text{prob}(\text{We say Yes})$$

$$= \text{prob}(i = \hat{j}) = \frac{1}{2}.$$

Signatures

€
1.04.08
(6)



Electronic world?

Remember public key crypto:

You have a private key.

Everybody knows your public key.
(can know)

Brain, nerves,
muscles

Example RSA signatures (ciphertext) & insecure

Lots of old
handwritings
& handwritten
signatures

Sign
Input: $x \in \mathbb{Z}_N$ document; (N, d) private key
Output: s signature

1. Compute $s := x^d \bmod \mathbb{Z}_N$.
2. Return s .

Verify
Input: x document, s signature, (N, e) public key.
Output: Yes, signature is valid. or NO!
1. Return ($x = s^e \bmod \mathbb{Z}_N$)

CORRECTNESS: $s^e = (x^d)^e = x^{ed} = x \quad \checkmark$

EFFICIENCY:

$O(n^4)$ for setup
 $\rightarrow O(n^3)$ for sign/verify.

$e \in$
 7.04.08
 (7)

SECURITY?

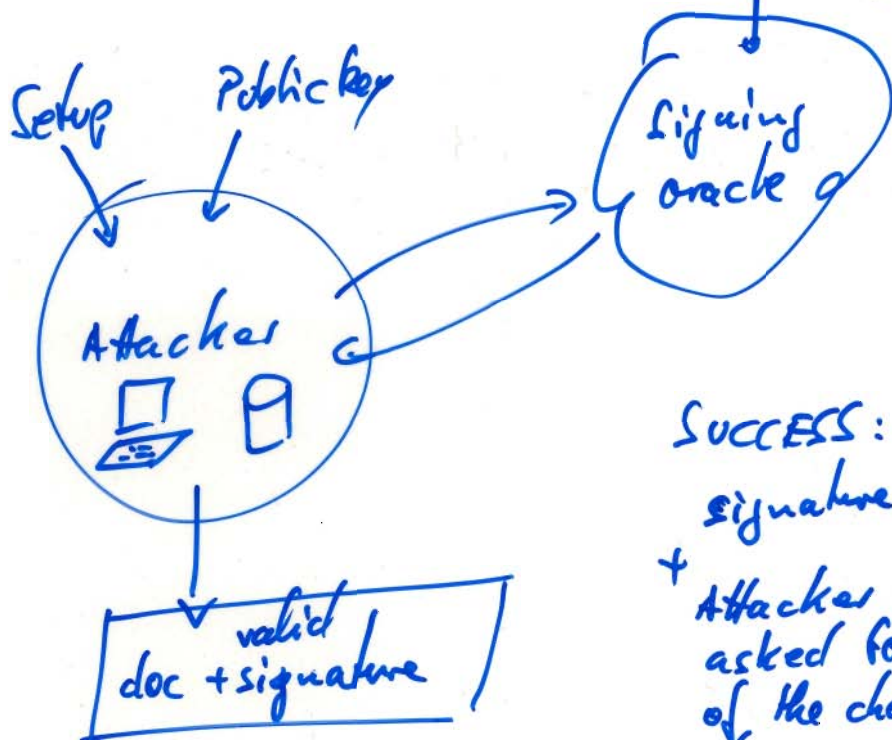
integrity?

NO:

$(x \cdot r^e, s \cdot r)$

fulfills $(x \cdot r^e) = (s \cdot r)^e$ ✓
 Private key

Attacker?



SUCCESS:
 signature valid
 + Attacker never asked for signature of the chosen document.

In our case:

Attacker 2

Input: (N, e, x)

Output: s a valid RSA signature on x

1. Write $x = x_1 \cdot x_2$ with $x_2 = r^e$

2. Ask signing oracle on x_1 : obtain s_1
 with $x_1 = s_1^e$ in \mathbb{Z}_N .

Now: $x = x_1 \cdot x_2 = s_1^e \cdot r^e = (s_1 \cdot r)^e$ in \mathbb{Z}_N

3. Return $(s_1 \cdot r)$

$\text{sign}(x_2) = r^e = r$

Tool: Hash functions

e€
1.04.08
②

$$h: \{0,1\}^* \rightarrow \{0,1\}^l$$

easy to evaluate. with l small number
standard for l : ~~128~~, 160, ... 512.

ONE-WAY
An attacker should not be able to invert this function, i.e. given a value $a \in \{0,1\}^l$ to find a document x with $h(x) = a$.

COLLISION-RESISTANT
An attacker should not be able to find two different documents $x_0 \neq x_1$ such that $h(x_0) = h(x_1)$.

Generic attack on ONE-WAY needs
expected 2^{e-1} steps.

Generic attack on COLL.-RESISTANT needs
expected $O(2^{e/2})$ steps.
 $\sqrt{2^e}$

SHA-1 : Find collision generically: 2^{80} op's.
(Secure Hash Algorithm #1) Huang & al (2004/5) : 2^{63} op's.

SHA2 family: 224, 256, 384, 512 bits.
{SHA512 is out library!}

How to increase the security
and have to sign large documents?

e€
1.04.08
(9)

Sign the hash value of the document
instead of the document itself.

Verify (RSA signature):

Input: x document, s signature, (N, e) public key.

Output: Yes/No.

1. Return

$$h(x) = s^e \text{ in } \mathbb{Z}_N.$$

Our old attack takes (x_1, s_1) valid signed
 (x_2, s_2) valid signed

& combines them $(x_1 x_2, s_1 s_2)$

but now that means:

$$h(x_1) = s_1^e$$

$$h(x_2) = s_2^e$$

and thus $h(x_1) h(x_2) = (s_1 s_2)^e$

usually: H

$$h(x_1 x_2)$$

Assume that B is a program that outputs
a-collision to the hash-function.

Task: construct an attacker on the
signature scheme.

Attacker on signature scheme

Input: setup, public key

Output: a doc with a valid signature

1. Ask B for two different documents x_1, x_2 with $h(x_1) = h(x_2)$.

2. Ask the signing oracle for a signature s on x_2 .

3. Return (x_1, s) .

So if B is successful so is this attacker.

Our security goal requires that there exists no successful attacker.

Thus such an attacker must also not exist.

I.e. we need that the hash function

is collision-resistant.

Assume that a given signature scheme fulfills our security goal, i.e. there exists no attacker as described above.

e€
2.04.08

(1)

Then the integrity of the document is granted.

PS otherwise somebody could change a document w/o making the signature invalid.
So this would be a successful attacker in our sense. \square

Then the identity of the signer is granted. \square

PS as above ... \triangle

Then the connection between document & signer is granted. \triangle

PS same again

For RSA signature with a full-domain hash function, i.e. every value in \mathbb{Z}_N is a possible hash value with same probability, we can prove that it is secure in the random oracle model.

El Gamel signatures / Schnorr signatures $e \in 204.08$
(2)

setup (a group G with generator g)

Verify Input: x document, $s = (b, r)$ signature

Output: Yes/No.

Verify Input: setup (a group G with generator g and a hash function $h: \{0,1\}^* \rightarrow \mathbb{Z}_{\#G}$ and a more or less trivial function $*$: $G \rightarrow \mathbb{Z}_{\#G}$ (structure less!))

public key (a group element $a \in G$)

document x ,

signature $s = (b, r)$.

Output: Yes/No.

1. Check input types:
 setup, public key ✓
 document $x \in \{0,1\}^*$,
 signature: $b \in G, r \in \mathbb{Z}_{\#G}$.

2. Check

$$(*) \quad a^{b^*} b^r = g^{h(x)} \text{ in } G.$$

Standard example for G :

$$G \leftarrow \mathbb{Z}_p^x, \quad g \in \mathbb{Z}_p^x \text{ of prime order } q \text{ (i.e. } g \neq 1, g^q = 1)$$

$$G = \langle g \rangle = \{1, g, g^2, \dots, g^{q-1}\}$$

So we determine q 160-bit prime,
 p 1024-bit prime with $q \mid p-1$
 (i.e. $p-1 \in \mathbb{Z}_q$)

An attacker needs to solve $(*)$.

€
2.04.08
(3)

Trial 1

Fix $b \in G$ somehow. ($b \neq 1$).

Then

$$\begin{array}{c} b^x \\ \uparrow \\ \text{known} \end{array} \overset{\text{unknown}}{\xrightarrow{x}} = \underbrace{g^{h(x)} a^{-b^x}}_{\text{known}} \text{ in } G.$$

This is DLP.

In other words: if DLP is easy
then solving $*$ (\equiv breaking the scheme)
is easy.

or: if breaking is difficult
then DLP must be difficult.

Trial 2

Fix $x \in \mathbb{Z}_{\#G}$ somehow.

Then the attacker needs to solve

$$a^{b^* b^x} = g^{h(x)} \text{ in } G.$$

Trial n

Impose another condition on (b, x)
(other than $b = \text{constant}$ or $x = \text{constant}$)

Trial n+1 Try to solve 1 equation for both variables
 \leadsto We do not know much.

ElGamal Setup:

e€
2.04.08

(4)

Fix a group G with a generator g
and known size $\#G$.

Fix a hash function $h: \{0,1\}^* \rightarrow \mathbb{Z}_{\#G}$.

Fix a structureless, very simple $*$: $G \rightarrow \mathbb{Z}_{\#G}$.
(almost surjective)

↑ In the standard example:

$$b \in G \subset \mathbb{Z}_p^*$$

so b is given by an integer B with $0 < B < p$.

then define $b^* = B \bmod q \in \mathbb{Z}_q = \mathbb{Z}_{\#G}$.

Toy example: $q=3, p=7$

$$b \in \mathbb{Z}_p^* \text{ say } b = 5 \bmod 7$$

$$\text{then } B = 5 \text{ so } b = B \bmod 7.$$

$$\text{then } b^* = 5 \bmod 3 = 2 \bmod 3.$$

$$\text{Beware: } b = 12 \bmod 7 = 5 \bmod 7$$

$$\text{but } 12 \bmod 3 \neq 2 \bmod 3.$$

Key setup

Choose $\alpha \in \mathbb{Z}_{\#G}$, compute $a = g^\alpha \in G$.

Private key $\alpha \in \mathbb{Z}_{\#G}$.

Public key $a \in G$.

ElGamal signing

Input: ~~x~~ document setup + private key,
x document

Output: $s = (b, \gamma)$ signature on x.

Idea: we solve the equation $\textcircled{1}$ by first taking the DL of it:

$$b^* \cdot \alpha + \gamma \cdot \log_g b = h(x) \text{ in } \mathbb{Z}_{\#G}$$

1. Choose $\beta \in_R \mathbb{Z}_{\#G}^*$ and
calculate $b = g^\beta$ in G .

2. Then solve

$$b^* \cdot \alpha + \gamma \cdot \beta = h(x) \text{ in } \mathbb{Z}_{\#G}.$$

ie. let $\gamma = \beta^{-1} \cdot (h(x) - b^* \alpha)$.

3. Return (b, γ) .

e€
2.04.08
(5)

Blind signatures

€
2.04.08
(6)



A blind signature is a signature on a document that you don't see.

Example

Blind RSA signature:

Customer

wants x to be signed.

Choose $\tau \in_R \mathbb{Z}_N$ at random.

Compute $X = x\tau^e$ in \mathbb{Z}_N .

~~(Not)~~

Bank

Has public key (N, e) .

$X \rightarrow$

$$Y = X^d \text{ in } \mathbb{Z}_N$$
$$[= X^{1/e} \text{ in } \mathbb{Z}_N]$$

$Y \leftarrow$

Compute $y = Y/\tau$.

Now y is a RSA signature on x :

$$y = Y/\tau = X^d/\tau = (x\tau^e)^d/\tau = x^d \underbrace{\tau^{ed}}_1/\tau$$
$$= x^d, \text{ so } \boxed{y^e = x}.$$

Why is that blind?

€
2.0408

7

Yes in one signing the bank saw X.

And later it sees a signed X. &

Then this could have been the one
actually signed if

$$r = (X/x)^d$$

So it's blind, even later bank cannot
make the link.

Thus we need a way to guarantee that
the x in question is something legal.

One solution to this is the "cut-and-choose
methodology":

The customer prepares
many (say 100)
envelopes and gives
them to the bank



Chance to cheat w/o detection:

$\frac{1}{100}$

The bank chooses
one and asks
the customer to
open all others

The bank checks
the others and if
all are ok, signs the chosen one.

e€
Duty List

Validity
checking

Unforgeability

Implementability

Anonymity

Untraceability

Double-spending
protection

Fairness

Reusability

Unframability

Spending
book keeping

Online/offline

Scalability

Interoperability

Atomicity

Acceptability

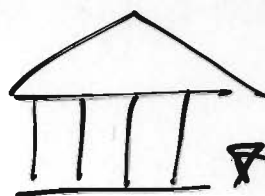
eg Open Source (Korach's
principle)

Economicity

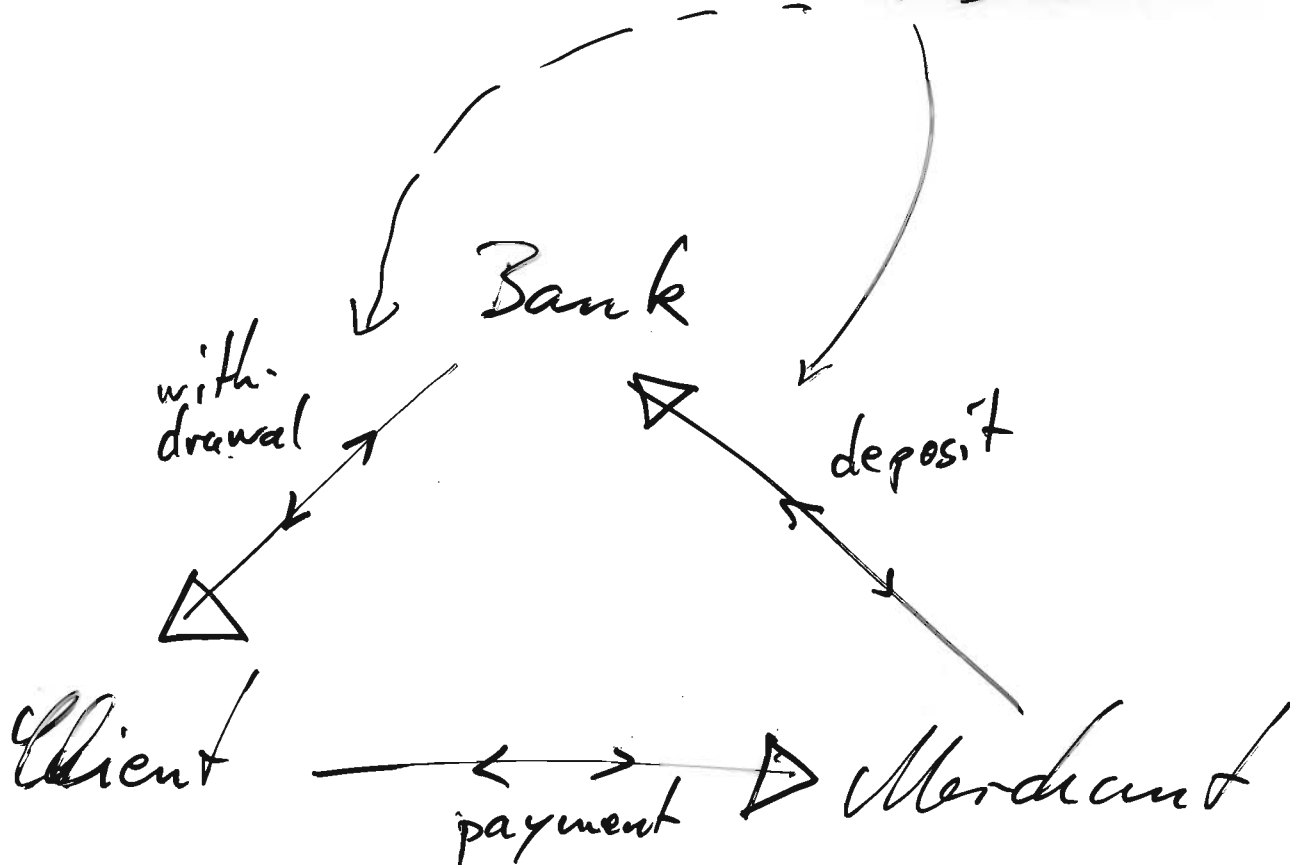
Setup
→ for bank signatures

e€
2.04.08

8



Trustee ...



"
some info
+ bank signature
on the entire or part of
the info.

might be random,
or hash of partially
(to the client) hidden
information

Double-spending protection:

usually the client id is hidden in the coin.

then merchant challenges the client

who in return sends partial info about his id.

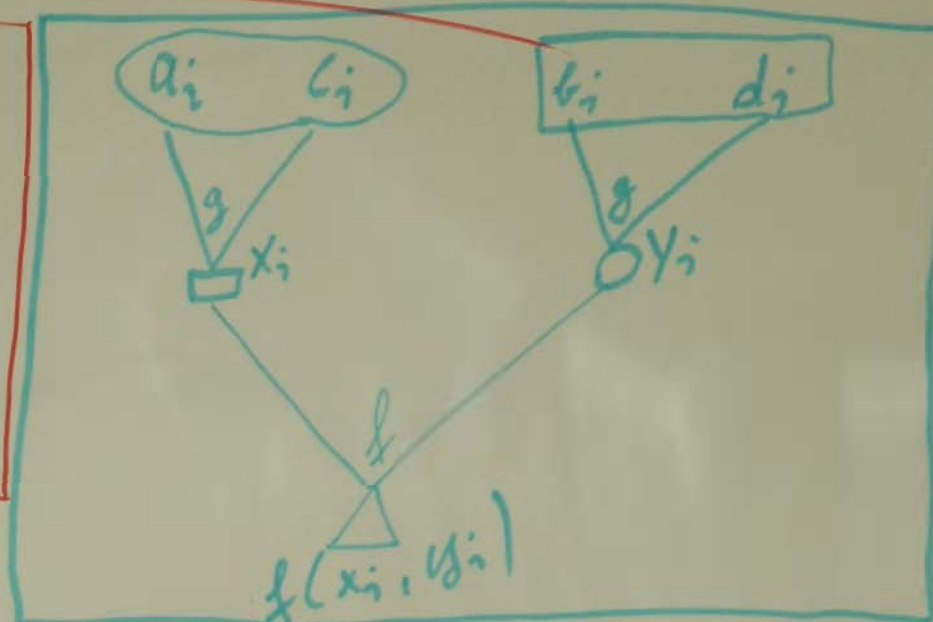
And such that if the client answers two challenges the id is complete.

Untraceable e-Cash

Chaum, Fiat & Naor (1989)

Needed

- RSA-Encryption (e, d)
- Hash-Funktionen f, g
- Blind signatures



Withdrawal

Alice

Bank

- ① Setup
- ② Prepare $B_i = r_i^e \cdot f(x_i, y_i)$ $i=1, \dots, k$
- ③ Choose half of B_i
- ④ Reveal parameters for selected B_i
- ⑤ Check selected B_i
Sign other B_i : B_i^d
Send ΠB_i^d
- ⑥ Extract Coin

Payment & Deposit

Alice

Bob

- ① Send Coin
- ② Send random bitstring
- ③ Send (x_i, b_i, d_i) or (a_i, c_i, y_i) according to Bob's bit string
- ④ Verify
- ⑤ Send coin and data for verification
- ⑥ Verify

Bank

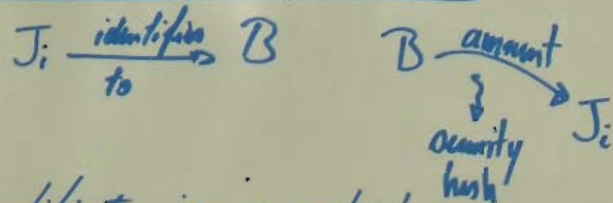
Offline Cash Transfer by Smart Cards

Brands (1994)

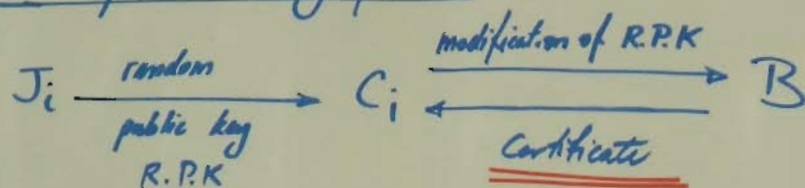
1. Opening an account

U (user) gets J_i (smart card) by B (bank)
 J_i has $\left\{ \begin{array}{l} \text{Private key + shared secret key with } B \\ \text{other hidden inf.} \end{array} \right.$

2. The withdrawal protocol

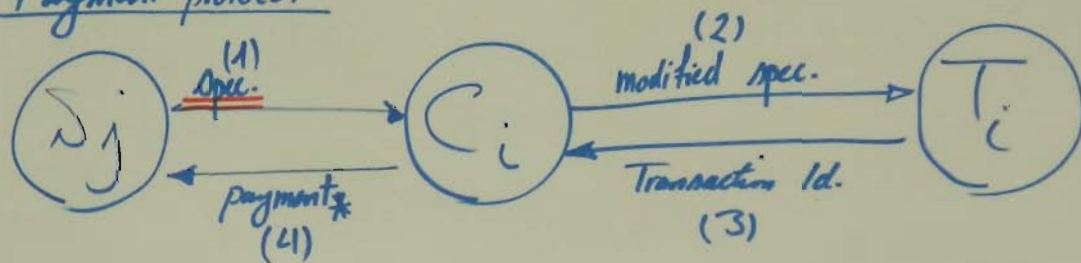


3. Certificate issuing protocol

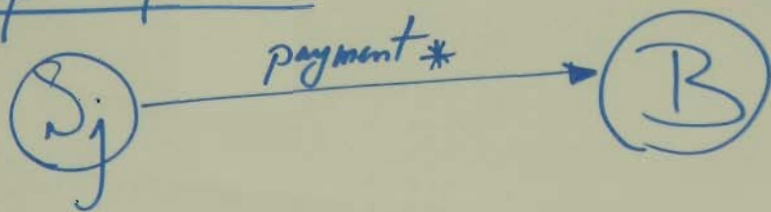


Double spending detection
Certificate: only once valid
specification: unique

4. Payment protocol



5. deposit protocol



pro
Anonymous

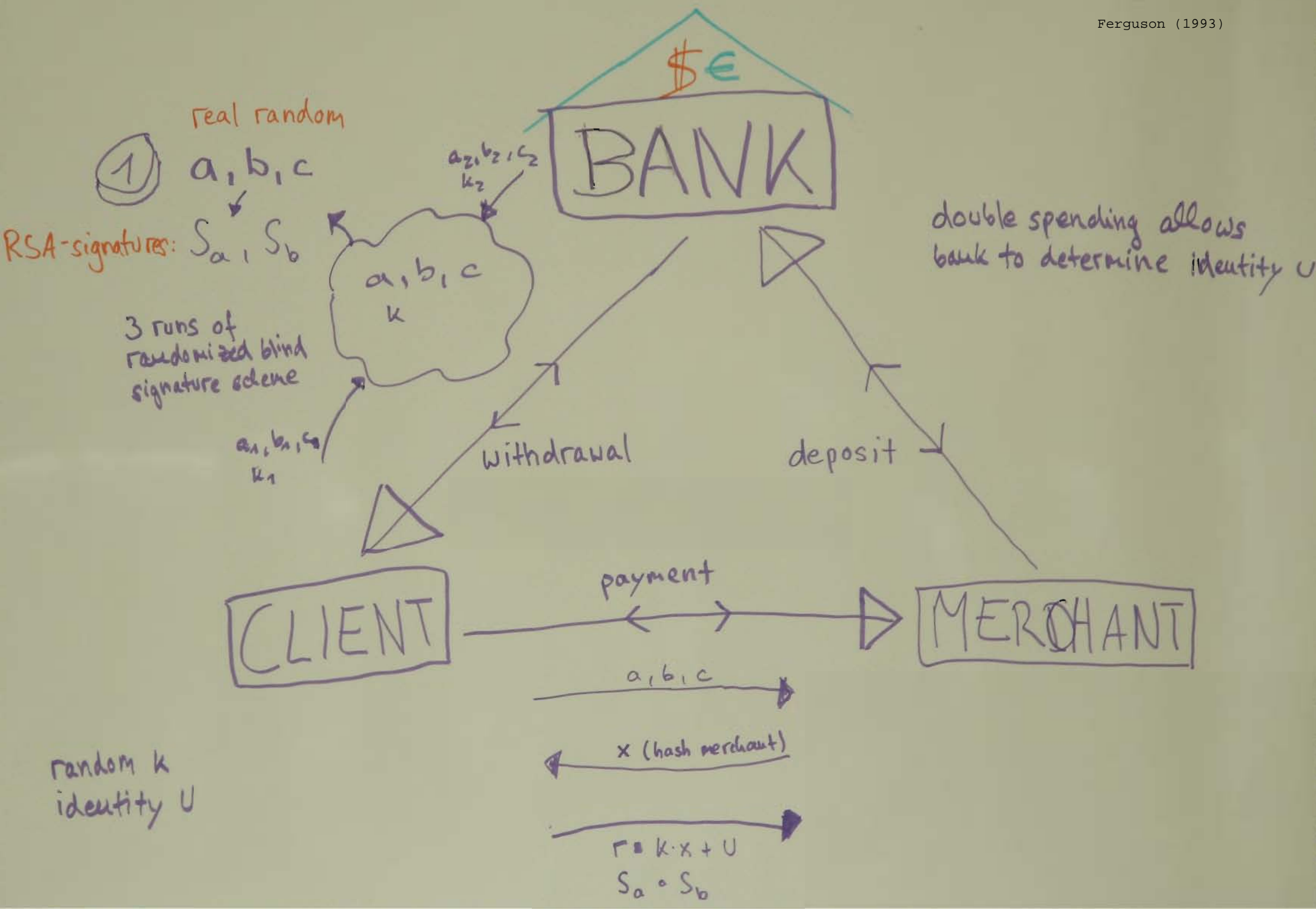
double double-spending

seems to be implementable

con
no fairness

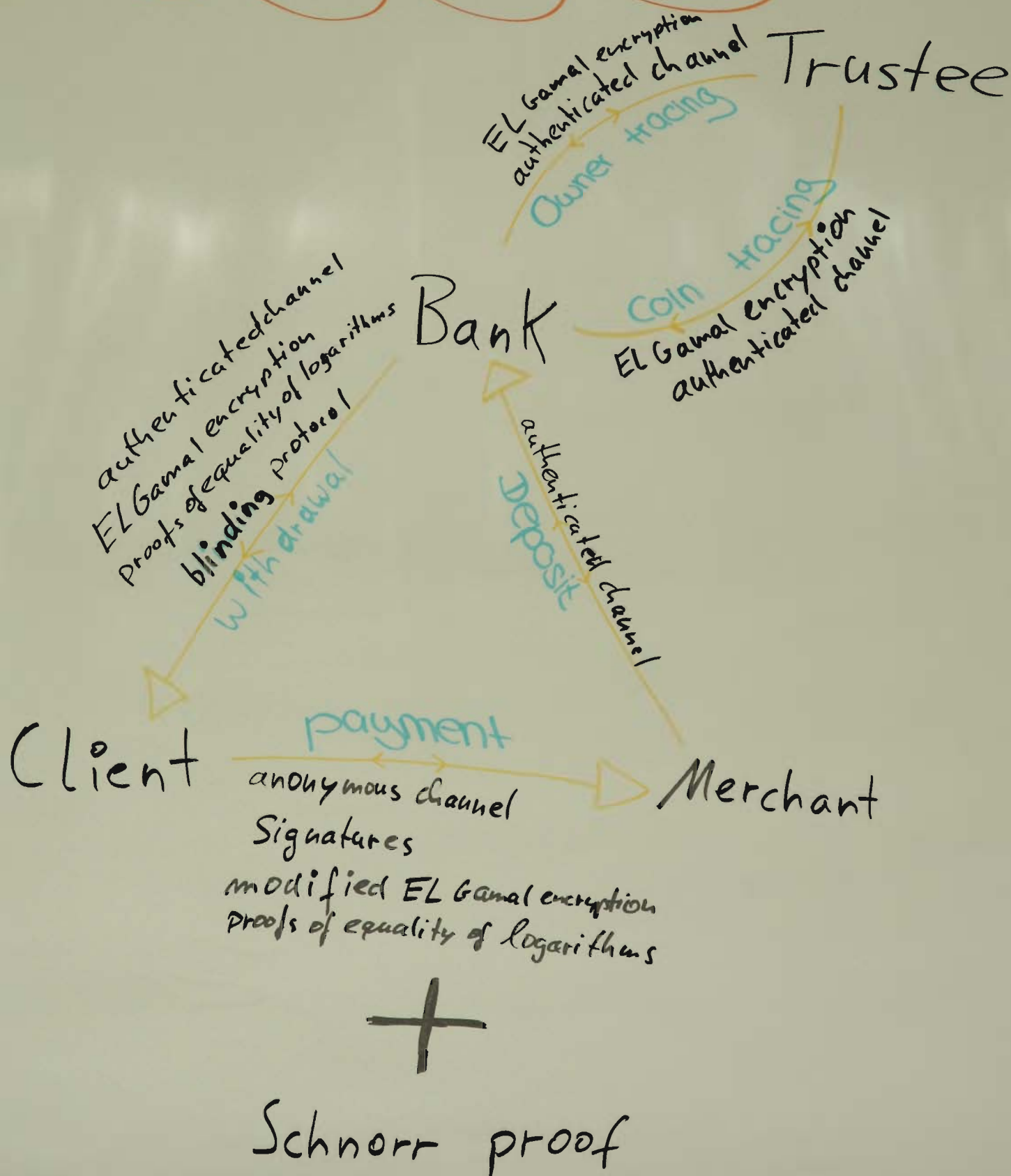
SINGLE TERM OFF-LINE COINS

Ferguson (1993)



Frankel, Tsiounis, Yung

(1996, 1998)

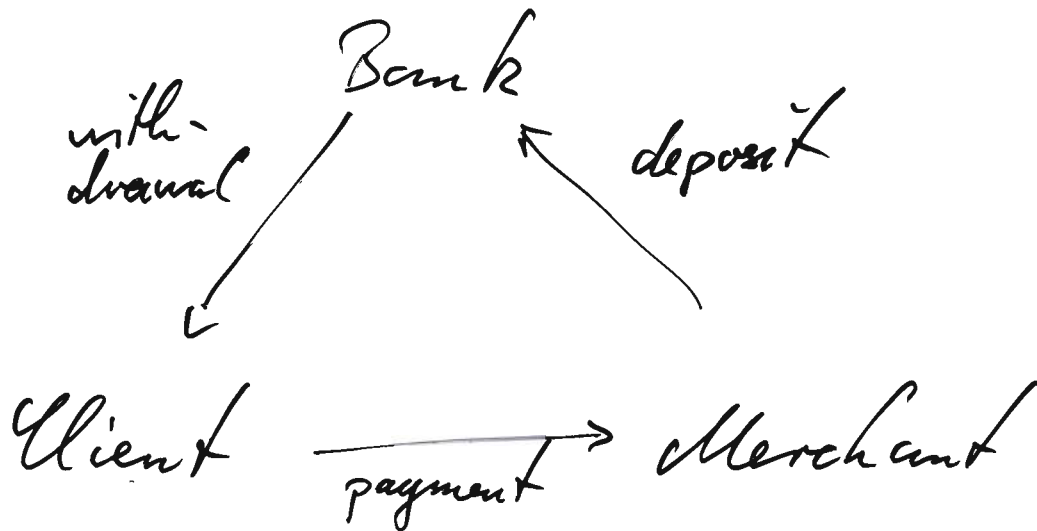


PR@: Fairness

Electronic cash system

Ferguson (1994)

e€
3.04.08
①



Double spending protection:

Chaum (?), Chaum, Fiat & Naor ('89)
→ cut-and-choose

Ferguson ('94)

uses secret-sharing:

Choose a polynomial $f \in \mathbb{F}_q[x]$
such that $f(0)$ is the secret
(encoded as an element of \mathbb{F}_q).

Hand out shares $(a, f(a))$.

As soon as $\deg f + 1$ share
come together the entire polynomial
and in particular the secret can be reconstructed.

For vs: $f(x) = k \cdot x + u$

$e \in$
3.0408
(2)

in the coin: g^k, g^u .

How to check that $(a, b) = (a, f(a))$?

$$\underline{(g^k)^a} \cdot \underline{g^u} = \underline{g^{f(a)}}.$$

Ferguson (1994)

This system is based on the difficulty of RSA and a discrete logarithm problem but it also uses some hash functions at sensitive places to (hopefully) increase the security. Further, polynomial secret sharing is used in order to decrease the coin size without loss of security. The important part here is Martin's challenge size, it must be large enough to prevent repetitions. The challenge size in Chaum *et al.* (1989) was $k/2$ bits, so the size of the coin grows linearly with the wanted challenge size. Here the challenge size depends only on the chosen group and is thus typically not much larger than with, say, $k = 4$. But let us first explain the polynomial secret sharing and the system.

1. Polynomial secret sharing

Suppose there is some secret x that we want to give to a group of people. Yet, the secret is very valuable and we do not trust a single person far enough to give him the secret. Think of the access code of the central safe of a bank or the start code of nuclear weapons. The solution is to distribute the secret: each person only gets part of the secret. Now, we know that to determine a polynomial f of degree less than k over some field \mathbb{F} we need to know k pairs $(x, f(x))$. By interpolation we can then recover f , in particular, say, $f(0)$. If we give one point $(x, f(x))$, $x \neq 0$, to each person then at least k of them must come together to recover the secret $f(0)$ and thus to be able to open the safe or to start the missile. Figure 1 shows a picture of a line over \mathbb{F}_{257} . Any two points determine the secret. But if we only know one point then any secret could complete the picture. In Figure 2 we see a line over \mathbb{F}_{256} , the elements of \mathbb{F}_{256} have been numbered in some systematical way for that purpose. Again any two points determine the line, one point could go with any secret. Figure 3 shows cubic curves. Only if we know at least four of its non-zero points then we can recover the secret.

2. The system

Following the description of the author we also first describe the payment thus specifying the form of the coins. For the payment process we then have to find a way of getting the appropriate blind signatures from the bank. The basic setup contains an RSA signature key pair of the bank with public key (N, v) . Additionally to the standard assumptions we require that v is a sufficiently large prime and that $\varphi(N)$ contains at least one large prime factor. Further some elements $g_1, g_2, g_3 \in \mathbb{Z}_N^\times$ of large order (minimal repetition length) are fixed. To be able to find them the bank should construct her primes p, q such that she knows large prime factors of $p - 1$ and $q - 1$. Next we need a suitable prime t such that $N \mid t - 1$ and elements $h_2, h_3 \in \mathbb{F}_t^\times$ of order N . Finally, the bank chooses hash functions $f_1: \mathbb{Z}_N^\times \rightarrow \mathbb{N}_{<v}$,

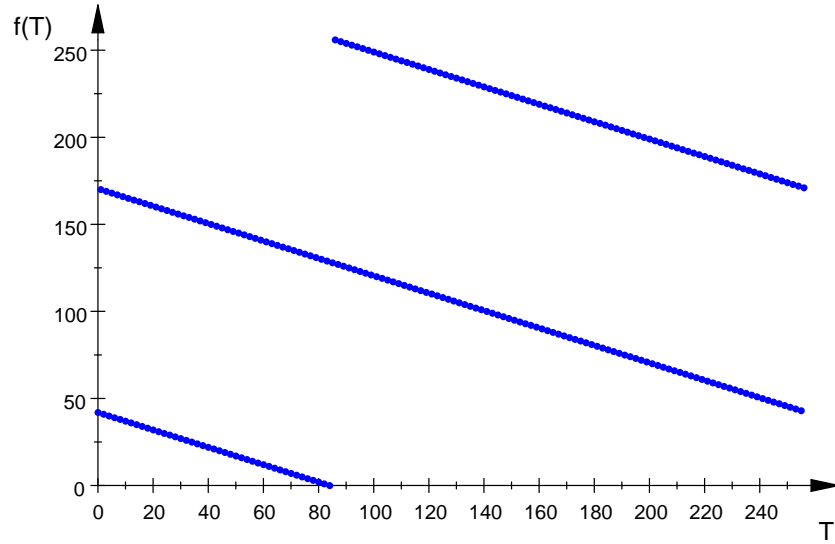


Figure 1: The line $f: \mathbb{F}_{257} \rightarrow \mathbb{F}_{257}$, $T \mapsto 128T + 42$ over the field \mathbb{F}_{257} carries the secret $f(0) \hat{=} 42$ and passes through zero at $T \hat{=} 84$. The elements of \mathbb{F}_{257} are represented as integers modulo 257 (which is prime!).

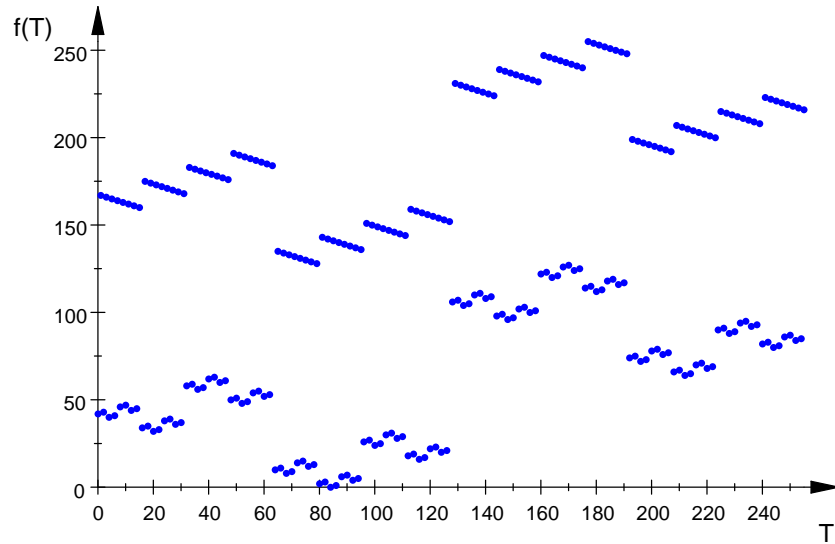


Figure 2: The line $f: \mathbb{F}_{256} \rightarrow \mathbb{F}_{256}$, $T \mapsto (x^7 + x^3 + x^2 + 1)T + (x^5 + x^3 + x)$ over the field \mathbb{F}_{256} carries the secret $f(0) = x^5 + x^3 + x \hat{=} 2^5 + 2^3 + 2 = 42$ and passes through zero at $T \hat{=} 84$. The elements of \mathbb{F}_{256} are represented as polynomials in x of degree less than 8 over $\mathbb{F}_2 = \mathbb{Z}_2$ modulo $x^8 + x^4 + x^3 + x + 1$ and identified with integers by ‘evaluating’ such a polynomial over the integers at $x = 2$.

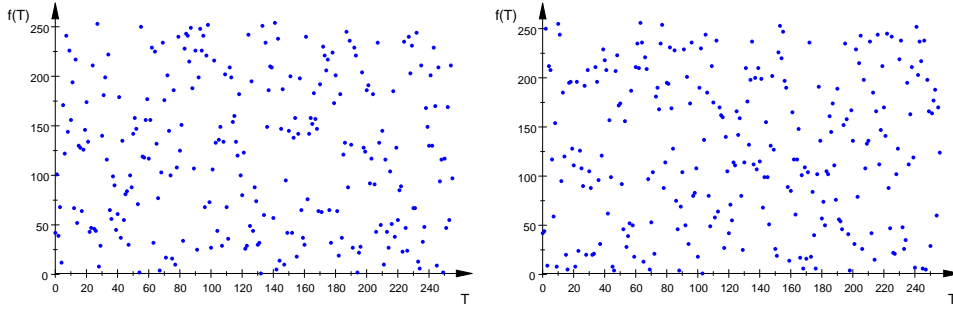


Figure 3: The cubic curve $f: \mathbb{F}_{256} \rightarrow \mathbb{F}_{256}$, $T \mapsto (x^7 + x^6 + x^5 + x^4 + x^3 + x + 1)T^3 + (x^7 + x^5 + x + 1)T^2 + (x^4 + x^2 + x + 1)T + (x^5 + x^3 + x)$ over \mathbb{F}_{256} on the left hand side and $f: \mathbb{F}_{257} \rightarrow \mathbb{F}_{257}$, $T \mapsto 20T^3 + 42T^2 + (-60)T + 42$ over \mathbb{F}_{257} on the right hand side each carry the secret $f(0) \hat{=} 42$. For our untrained eyes the nice structure of this curve is not visible but still: any four points determine the entire polynomial and thus the secret.

$f_2, f_3: \mathbb{F}_t^\times \rightarrow \mathbb{N}_{<v}$, and $f_4: \mathbb{N}_{<v} \times \mathbb{N}_{<v} \rightarrow \mathbb{Z}_N^\times$. The bank publishes the data

$$(N, v, g_1, g_2, g_3, t, h_2, h_3, f_1, f_2, f_3, f_4).$$

Further Alice' identity is coded in a value $U \in \mathbb{N}_{<v}$.

PROTOCOL 1. Bank setup.

1. The bank chooses an RSA key pair: Find p, q primes such that $N = p \cdot q$ has, say, 1024 bits. Compute $L = (p - 1)(q - 1)$ and choose a suitably large, say 128 bit, prime $v \in \mathbb{N}_{<L}$ coprime to L . Calculate $(1/v) = v^{-1} \bmod L$.
2. Find elements $g_1, g_2, g_3 \in \mathbb{Z}_N^\times$ of large order. This might be a problem since the bank must know the factorization of L to determine the order of a randomly chosen element. But p and q can be constructed such that at least a large prime factor P of $p - 1$ and Q of $q - 1$ is known. Then $x^{\frac{L}{PQ}}$ is an element of order 1, P , Q , or PQ for any $x \in \mathbb{Z}_N^\times$ and elements of order PQ can be found by repeating this some times.
3. Find a prime t with $t \equiv_N 1$. (Since the size of t needs only be of the same order as N , she might simply choose the smallest prime of the form $xN + 1$.)
4. Find elements $h_2, h_3 \in \mathbb{F}_t^\times$ of order N . Again this is easy by testing the order of $x^{\frac{t-1}{N}}$ for randomly chosen $x \in \mathbb{F}_t^\times$ which can be only 1, p , q , or $N = pq$.
5. Fix hash functions $f_1: \mathbb{Z}_N^\times \rightarrow \mathbb{N}_{<v}$, $f_2, f_3: \mathbb{F}_t^\times \rightarrow \mathbb{N}_{<v}$, and $f_4: \mathbb{N}_{<v} \times \mathbb{N}_{<v} \rightarrow \mathbb{N}_{<v}$.
6. Publish

$$(N, v, g_1, g_2, g_3, h_2, h_3, f_1, f_2, f_3, f_4).$$

The only extra information the bank needs is its secret exponent $(1/v)$.

Note that we will do a lot of calculations in the RSA domain \mathbb{Z}_N^\times but some calculations also will take place in the field \mathbb{F}_t .

The coin consists of randomly chosen values $a, b, c \in \mathbb{Z}_N^\times$ from which anybody can compute $A = ag_1^{f_1(a)}$, $B = bg_2^{f_2(h_2^b)}$, $C = cg_3^{f_3(h_3^c)}$. Further a random parameter $k \in \mathbb{N}_{<v}$ and signatures $S_1 = (AC^k)^{(1/v)}$ and $S_2 = (BC^U)^{(1/v)}$ are part of the coin.

PROTOCOL 2. Payment.

1. Alice hands over (a, b, c) to Martin. $\xrightarrow{(a, b, c)}$
2. Martin chooses a random challenge $x \in \mathbb{N}_{<v}$. \xleftarrow{x}
3. Alice computes $r + \hat{r}v \leftarrow kx + U$ with $r \in \mathbb{N}_{<v}$ and a signature R to $A^x BC^r$ by $R \leftarrow S_1^x S_2 C^{-\hat{r}} = (A^x BC^r)^{(1/v)}$. She sends (r, R) to Martin. $\xrightarrow{(r, R)}$
4. Martin verifies that the signature is valid: all transmitted data are in the required domains and

$$R^v \stackrel{?}{=} A^x BC^r.$$

Note that he can do that.

Depositing the coin is easy, too:

PROTOCOL 3. Deposit.

1. Martin sends the entire transcript of the payment Protocol 1 to the bank. $\xrightarrow{(a, b, c, x, r, R)}$
2. She then looks up the signature in her database.
 - If she does not find it, Martin gets his money put on his account and a receipt.
 - Otherwise, the bank detects a double spending just as in the other systems:
 - If the challenges x and x' are also equal then Martin has tried to redeposit a coin.
 - Otherwise the bank tries to reveal Alice' identity. For now the bank knows $r \equiv_v kx + U$ and $r' \equiv_v kx' + U$ modulo v which is just a linear system of equations for k and U . Now she can take Alice to court for double spending.

There are several points to be taken into account for the withdrawal process. Of course the first requirement is that the bank cannot link the withdrawal and the deposit of a coin (unless a double spending occurs). Further, it shall be guaranteed that the parameters a, b, c and k are chosen randomly. Both parties, in particular the bank in our case, have to be sure that these parameters are not 'made up'. To do so Alice and the bank each choose a part, say a' and a'' of these parameters and at the end they take the product $a = a'a''$. Only both must make their choice independently whereas we have no way of guaranteeing a parallel transmission of the respective shares. (Actually, this seems very similar to 'Coin flipping by phone', Blum 1982.)

To achieve this, Alice first chooses a' and then transmits some information \tilde{A} which binds her to this value of a' . Then the bank chooses a'' and sends it to Alice. Actually, in our case the product must only be known to Alice. To make sure that Alice continues as desired, Alice sends something which requires that she uses the bank's a'' in order to give her the desired meaningful signature. Or the bank's answer depends on the information \tilde{A} that binds Alice. Then the answer is only useful to Alice if she sticks to her previously chosen value a' .

3. Randomized blind signatures

First we consider how to get a *randomized blind signature*. Randomized means that the bank will be sure that the used parameter was indeed chosen at random. Blind means, as usual, that the bank cannot link the final signature to the transcript of the signature protocol. And of course Alice should not be able to generate such a signature on her own (this makes it a signature). Thus this scheme will be well suited for our needs. Ferguson attributes it to Chaum (1992). Additionally we use a one-way hash function $f: \mathbb{Z}_N^\times \rightarrow \mathbb{N}_{<v}$.

PROTOCOL 4. Randomized blind signature.

1. Alice randomly chooses $a', \alpha \in \mathbb{Z}_N^\times$ and $\sigma \in \mathbb{N}_{<v}$. She computes $\tilde{A} \leftarrow \alpha^v a' g^\sigma$ and sends that to the bank. $\xrightarrow{\tilde{A}}$
2. The bank randomly chooses $a'' \in \mathbb{Z}_N^\times$ and sends it to Alice. $\xleftarrow{a''}$
3. Alice computes $a \leftarrow a' a'' \in \mathbb{Z}_N^\times$ and an adjusting exponent $e + \hat{e}v \leftarrow f(a) - \sigma$ with $e \in \mathbb{N}_{<v}$ and sends e to the bank. \xrightarrow{e}
4. The bank computes $\tilde{A} \leftarrow \tilde{A} \cdot a'' g^e$ and sends Alice a signature $\tilde{S} \leftarrow \tilde{A}^{(1/v)}$ of it. $\xleftarrow{\tilde{S}}$
5. Alice unblinds the signature to obtain $S \leftarrow \tilde{S} \alpha^{-1} g^{\hat{e}}$. Now she has a signature pair (a, S) satisfying

$$(5) \quad S^v \stackrel{?}{=} a g^{f(a)}.$$

Before we discuss attacks let us have a short glance at the correctness. There is one complication that we did not mention in advance. Actually, Alice must hand over $e \in \mathbb{N}_{<v}$ instead of $e + \hat{e}v$ in order to keep her secrets protected. Unfortunately, it is not allowed to calculate modulo v (or any other number Alice knows of) in the exponent of g . She only knows that g has large order but she has no idea which one. Thus she will obtain a v -th root of $a g^{f(a) - \hat{e}v}$ instead of a v -th root of $a g^{f(a)}$. Luckily this is correctable since the deviation is a v -th power of a known value. Indeed, we

have

$$\begin{aligned}
S^v &= \tilde{S}^v \alpha^{-v} g^{\hat{e}v} \\
&= \tilde{A} \alpha^{-v} g^{\hat{e}v} \\
&= \tilde{A} \cdot a'' g^e \alpha^{-v} g^{\hat{e}v} \\
&= \alpha^v a' g^\sigma \cdot a'' \alpha^{-v} g^{f(a)-\sigma} \\
&= a g^{f(a)}.
\end{aligned}$$

First, note the relations between the values in the transcript: Clearly, Step 4 in Protocol 3 implies

$$(6) \quad \tilde{S}^v = \tilde{A} \cdot a'' g^e.$$

Everything else in the transcript is independent, as we will see shortly. Indeed, even if Alice follows the protocol any combination of \tilde{A} , a'' and e can occur: First choose any value for a , then solve $e + \hat{e}v = f(a) - \sigma$ for $\sigma \in \mathbb{N}_{<v}$ and \hat{e} , $a = a'a''$ for a'' , and $\tilde{A} = \alpha^v a' g^\sigma$ for α . (We do not care for efficiency here!) Thus (5) is the only relation. Each protocol transcript even occurs with the same probability. The only choice is the choice of a , all other solutions are unique. Thus in order to obtain a valid signature from the protocol Alice can choose \tilde{A} and e but must then go along with a'' and \tilde{S} as given by the bank. Though Alice can choose \tilde{A} as a v -th power of something she knows, her major problem is that she does not know the v -th root of a'' and thus cannot correct this factor to her needs without breaking RSA.

What if the bank tries to trace Alice? Can she get any information on the pair (a, S) that is Alice's signature at the end? No, she cannot. Indeed, each such pair occurs with the same probability from the view of the bank. The bank knows \tilde{A} , a'' , e and \tilde{S} . Suppose Alice gets (a, S) . Then there is exactly one choice for Alice that can have produced this outcome: $\sigma \in \mathbb{N}_{<v}$ and \hat{e} are uniquely determined by $e + \hat{e}v = f(a) - \sigma$, α by $S = \tilde{S} \alpha^{-1} g^{\hat{e}}$, and a' by $a = a'a''$. The equation $\tilde{A} = \alpha^v a' g^\sigma$ is implied by (5): $\tilde{A} = \tilde{S}^v \cdot (a'')^{-1} g^{-e} = \alpha^v S^v g^{\sigma-f(a)} / a'' = \alpha^v a g^{f(a)} g^{\sigma-f(a)} / a'' = \alpha^v a' g^\sigma$.

Let us see what happens if Alice tries to cheat. Clearly, she cannot solve (4) after fixing a unless she breaks the bank's signature which is assumed to be infeasible. But can she use the signature generation with a more or less prescribed a ? As already stated only (5) binds the values of the transcript. Suppose she wants to get along with a prescribed a . What would she have to do in order to get a signature for it? To satisfy (5) she must solve $a g^{f(a)} = \tilde{A} \cdot a'' g^e$ for e . She can choose \tilde{A} in a clever way, yet only before she knows a'' . Writing $e + \hat{e}v = f(a) - \sigma$ the equation $a = \tilde{A} \cdot a'' g^{-\sigma-\hat{e}v}$ must be solved for σ . Actually no matter how she has chosen \tilde{A} the task is to compute a discrete logarithm. But of course the parameters will be adjusted such that computing a discrete logarithm with base g is not feasible. By trying several σ at random she might get control of some bits of a but no more. Thus there seems at least to be no obvious way for Alice to cheat.

If Alice tries to use some more of the structure she might try to use some multiple of a power of (5) to obtain a valid signature on some expression $ag^{f(a)}$:

$$D\tilde{S}^{Ev} = D\tilde{A}^E(a'')^E \cdot g^{eE}$$

First note that D can only help if Alice knows a v -th root but that does not lead her far. To be helpful she might try to adjust this such that

$$\begin{aligned} (\tilde{A}a'')^E &= ag^t, \\ t + eE &= f(a) \end{aligned}$$

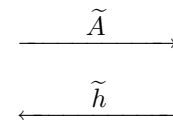
with some $t \in \mathbb{N}_{<v}$. Alice can use the first equation only after she knows a'' , when \tilde{A} is already fixed. So the obvious way to solve these equations is to choose E and t and determine a by the first equation. The control over a she can obtain this way depends on her ability of computing discrete logarithms with respect to g or $\tilde{A}a''$. Finally, the second equation determines e . However, that means that A , E and t must be chosen before e is transmitted.

Note that computing a discrete logarithm with base $\tilde{A}a''$ might be feasible! If the order of $\tilde{A}a''$ is smooth and can be determined efficiently then we can compute discrete logarithms efficiently and thus find a ‘good’ E . So we choose a , compute E and e . The order of the group \mathbb{Z}_N^\times however is unknown to Alice and infeasible to find (unless she breaks RSA). The bank could adjust a'' a little to avoid very low order elements. Yet, this affects the distribution of a'' and might not be desirable. Probably, it is true anyway that most elements of \mathbb{Z}_N^\times are difficult discrete logarithm bases provided $\varphi(N)$ contains large prime factors.

A way to stop Alice from even trying the just described manipulation is to change the scheme a little. In the previous ‘attack’, it was essential that Alice can compute $(a'')^E$. If we replace a'' by $h^{a''}$ then Alice cannot simply compute the corresponding $h^{((a'')^E)}$ from $h^{a''}$. Since we compute $a = a'a''$ in \mathbb{Z}_N the order of h must divide N . But we are not bound to the domains already in use and simply choose a prime t such that $h \in \mathbb{F}_t$ of order N exists, that is $t = \rho N + 1$ for some $\rho \in \mathbb{N}$. Once t is found any element $x \in \mathbb{F}_t^\times$ raised to the power $\frac{t-1}{N}$ gives an element $h = x^{\frac{t-1}{N}}$ of order 1, p , q , or N . The bank can easily exclude the first three cases by checking $h \neq 1$, $h^p \neq 1$ and $h^q \neq 1$. A drawback of this is that Alice cannot verify that. She is only able to check $h^N = 1$ and $h \neq 1$. But this is not really severe because it is in the bank’s interest to have an element of highest possible order there. Of course we now have to modify the definition of the hash function, we need $f: \mathbb{F}_t^\times \rightarrow \mathbb{N}_{<v}$. In total we have the following

PROTOCOL 7. Randomized blind signature without exponential attack.

1. Alice randomly chooses $a', \alpha \in \mathbb{Z}_N^*$ and $\sigma \in \mathbb{N}_{<v}$. She computes $\tilde{A} \leftarrow \alpha^v a' g^\sigma$ and sends that to the bank.
2. The bank randomly chooses $a'' \in \mathbb{Z}_N^\times$, computes $\tilde{h} \leftarrow h^{a''}$ and sends it to Alice.



3. Alice computes an adjusting exponent $e + \hat{e}v \leftarrow f(\tilde{h}^{a'}) - \sigma$ with $e \in \mathbb{N}_{<v}$ and sends it to the bank.
4. The bank computes $\bar{A} \leftarrow \tilde{A} \cdot a''g^e$ and sends Alice a signature $\tilde{S} \leftarrow \bar{A}^{(1/v)}$ of it along with a'' .
5. Alice calculates $a \leftarrow a'a''$ and unblinds the signature to obtain $S \leftarrow \tilde{S}\alpha^{-1}g^{\hat{e}}$. Now she has a signature pair (a, S) satisfying

$$(8) \quad S^v \stackrel{?}{=} ag^{f(h^a)}.$$

OPEN QUESTION 9. *Could Alice in either variant obtain more signatures than the number of times she executes the protocol?*

4. Withdrawal

For the withdrawal process we will use the previous signature scheme three times in parallel. Actually, for signing a we use the simple version and for signing b and c we use the one which is protected against the exponential attack. The first will be protected by an additional factor derived from the other two. As in the above protocols, Alice and the bank will each choose a share of the three values. Yet, a will be furthermore linked to the other two. This procedure would hand over three signatures to Alice. As we already saw in Protocol 1 which defined the payment from Alice to Martin, Alice needs signatures of AB^k and AC^U . Since we are using the RSA scheme to compute signatures these two are merely combinations of the three signatures to A , B and C .

The bank must be sure that U is used as specified since this is the identity coded into the coin. It will enable the bank to trace Alice in case of a double spending. This will be guaranteed since the bank puts together the second signature as one for AC^U .

It is in Alice' interest that k is randomly chosen and only known to herself since this parameter protects her identity! If it were known to anyone else then after only one payment U could be computed. But also the bank shall be sure that this parameter is chosen at random because otherwise Alice could try to fit this parameter according to her needs. Thus the bank will only hand over a signature to $A^{1/k'}C^{k''}$ without any knowledge of k' but with almighty power over k'' . Using $A^{1/k'}$ (implicitly) is made possible by choosing a as a k' -th power. Alice can later raise the result to the k' -th power and thus giving her a signature of $AC^{k'k''}$ as desired.

One problem arises again several times: Alice has to correct the exponents that shall be dealt with only modulo v . For example, this happens to $k'k''$. The final exponent to be used must be $k = (k'k'') \bmod v$. Since the difference is a multiple of v in some exponent Alice can correct that even in the v -th root. As can be verified in the protocol the corrections \hat{e}_2 and \hat{e}_3 are either 0 or -1 . But the corrections $\hat{1}$, \hat{e}_1 , and \hat{k} use the entire range $\mathbb{N}_{<v}$.

PROTOCOL 10. Withdrawal.

1. Alice chooses random shares $a', b', c' \in_R \mathbb{Z}_N^\times$, random blinding bases $\alpha, \beta, \gamma \in_R \mathbb{Z}_N^\times$, and random blinding exponents $\sigma, \tau, \varphi \in_R \mathbb{N}_{<v}$. She computes the blinded candidates $\tilde{A} \leftarrow \alpha^v a' \cdot g_1^\sigma$, $\tilde{B} \leftarrow \beta^v b' \cdot g_2^\tau$, $\tilde{C} \leftarrow \gamma^v c' \cdot g_3^\varphi$ and sends them to the bank.
2. The bank chooses her random shares $a'', b'', c'' \in_R \mathbb{Z}_N^\times$ and sends $a'', \tilde{h}_2 \leftarrow h_2^{b''}, \tilde{h}_3 \leftarrow h_3^{c''}$ to Alice.
3. Alice computes

$$\begin{aligned} e_2 + \hat{e}_2 v &\leftarrow f_2(\tilde{h}_2^{b'}) - \tau && \text{with } e_2 \in \mathbb{N}_{<v}, \\ e_3 + \hat{e}_3 v &\leftarrow f_3(\tilde{h}_3^{c'}) - \varphi && \text{with } e_3 \in \mathbb{N}_{<v}. \end{aligned}$$

and chooses $k' \in_R \mathbb{N}_{<v}^\times$. After computing $a \leftarrow (a' a'' \cdot f_4(e_2, e_3))^{k'}$ and $k^- \in \mathbb{N}_{<v}$, $\hat{1} \in \mathbb{N}$ such that $k' k^- = 1 + \hat{1}v$, she computes

$$e_1 + \hat{e}_1 v \leftarrow k^- f_1(a) - \sigma \quad \text{with } e_1 \in \mathbb{N}_{<v}.$$

Then she sends the exponents (e_1, e_2, e_3) to the bank.

4. The bank computes $\overline{A} \leftarrow \tilde{A} a'' f_4(e_2, e_3) g_1^{e_1}$, $\overline{B} \leftarrow \tilde{B} b'' g_2^{e_2}$, $\overline{C} \leftarrow \tilde{C} c'' g_3^{e_3}$. Then the bank chooses her share $k'' \in_R \mathbb{N}_{<v}^\times$ of k . She then computes the signatures $\tilde{S}_1 \leftarrow (\overline{A} \overline{C}^{k''})^{(1/v)}$ and $\tilde{S}_2 \leftarrow (\overline{B} \overline{C}^U)^{(1/v)}$ and sends them to Alice.
5. Alice puts everything together: she computes $b \leftarrow b' b'', c \leftarrow c' c''$ in \mathbb{Z}_N^\times , and $k + \hat{k}v \leftarrow k' k''$ with $k \in \mathbb{N}_{<v}$. Now she can compute

$$A \leftarrow a g_1^{f_1(a)}, \quad B \leftarrow b g_2^{f_2(h_2^b)}, \quad C \leftarrow c g_3^{f_3(h_3^c)}$$

and unblind the signatures $S_1 \leftarrow \left(\tilde{S}_1 \left(\alpha^{-1} g_1^{\hat{e}_1} \right) \left(\gamma^{-1} g_3^{\hat{e}_3} \right)^{k''} \right)^{k'} g_1^{-f_1(a) \hat{1}} C^{-\hat{k}}$ and $S_2 \leftarrow \tilde{S}_2 \left(\beta^{-1} g_2^{\hat{e}_2} \right) \left(\gamma^{-1} g_3^{\hat{e}_3} \right)^U$. She now has a coin (a, b, c, k, S_1, S_2) with the property

$$(11) \quad S_1^v = AC^k, \quad S_2^v = BC^U.$$

First we verify that this indeed fulfills the claimed equations (10). What the bank obtains actually is

$$\begin{aligned} \overline{A}^{k'} &= A \cdot \left(\left(\alpha g_1^{-\hat{e}_1} \right)^{k'} g_1^{f_1(a) \hat{1}} \right)^v, \\ \overline{B} &= B \cdot (\beta g_2^{-\hat{e}_2})^v, \\ \overline{C} &= C \cdot (\gamma g_3^{-\hat{e}_3})^v. \end{aligned}$$

With this information we can exploit the definitions:

$$\begin{aligned}
S_1^v &= \left(\left(\tilde{S}_1 \left(\alpha^{-1} g_1^{\hat{e}_1} \right) \left(\gamma^{-1} g_3^{\hat{e}_3} \right)^{k''} \right)^{k'} g_1^{-f_1(a)\hat{1}} C^{-\hat{k}} \right)^v \\
&= \overline{A}^{k'} \left(\left(\alpha^{-1} g_1^{\hat{e}_1} \right)^{k'} g_1^{-f_1(a)\hat{1}} \right)^v \left(\overline{C} \left(\gamma^{-1} g_3^{\hat{e}_3} \right)^v \right)^{k'k''} C^{-\hat{k}v} \\
&= AC^{k'k''-\hat{k}v} = AC^k
\end{aligned}$$

and similarly

$$\begin{aligned}
S_2^v &= \left(\tilde{S}_2 \left(\beta^{-1} g_2^{\hat{e}_2} \right) \left(\gamma^{-1} g_3^{\hat{e}_3} \right)^U \right)^v \\
&= \overline{B} \left(\beta^{-1} g_2^{\hat{e}_2} \right)^v \left(\overline{C} \left(\gamma^{-1} g_3^{\hat{e}_3} \right)^v \right)^U \\
&= BC^U.
\end{aligned}$$

Thus the key equations (10) hold.

In order to prevent the bank from framing an innocent Alice for double spending at some time Alice must provide a signature for this identity U . If this is not the case not only Alice will not trust the system but also the bank will not be able to prosecute Alice for a potential double spending. No court would blame Alice if she *can* be framed by the bank. Yet, we must somehow guarantee this in the withdrawal process, see below. The first thought how to implement this is to make U a signed version of Alice' identity. But then the bank cannot directly control that U has the correct form and thus Ferguson suggests a different approach.

5. Summary

Ferguson (1994) uses polynomial secret sharing to allow many possible queries. To embed a polynomial $kx + U$ into the system we proceed like this: Three numbers a, b, c are chosen at random by the bank and Alice. For the mutual security it is important that each partner is sure that these figures are indeed random. This is done by something similar to 'coin flipping by phone'. Alice and the bank each choose a part of each number and the actual number then is composed of these two parts. Yet only Alice will know the outcome of the random number. This makes the system anonymous. From these numbers are derived three numbers A, B, C with the help of some one-way functions. This ensures that Alice has almost no influence on the specific values of these three numbers. In the withdrawal process the bank sends Alice RSA signatures for AC^k and BC^U . Here also k must be a random quantity and again both must be sure of it.

To answer a query x by Martin Alice shows a signature R to $(AC^k)^x(BC^U) = A^x BC^{kx+U}$. She can produce this new signature from the two she knows. Clearly, she must also hand over $r = kx + U$ since Martin cannot compute this quantity. If the bank gets two such answers the bank can solve for k and U and thus reveal Alice' identity coded in U .

That is a very brief sketch of the system. There are some complications in the way the numbers a, b, c and k are chosen and some technical details that are used to prevent certain kinds of attacks.

References

MANUEL BLUM (1982). Coin flipping by telephone. In *CRYPTO 1981*, 133–137. IEEE. ISSN 0000-0133. URL <http://www-2.cs.cmu.edu/~mblum/research/pdf/coin/>.

DAVID CHAUM (1992). Randomized blind signature. Personal communication with Niels Ferguson.

DAVID CHAUM, AMOS FIAT & MONI NAOR (1989). Untraceable Electronic Cash (Extended Abstract). In *Advances in Cryptology: Proceedings of CRYPTO '88*, Santa Barbara CA. URL <http://citeseer.nj.nec.com/chaum89untraceable.html>.

NIELS FERGUSON (1994). Single Term Off-Line Coins. In *Advances in Cryptology: Proceedings of EUROCRYPT 1993*, Lofthus, Norway, TOR HELLESETH, editor, number 765 in Lecture Notes in Computer Science, 318–328. Springer-Verlag, Heidelberg. ISBN 3-540-57600-2. ISSN 0302-9743. URL <http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=765&spage=318>.

```
/* Returns whether the given signature is valid */
ferguson::is_valid_signature := proc(a, b, c, x, r, R)

begin

    // Get the bank's public key
    read( "X:\\io\\bank_public.mu" );

    // Check data types
    if( a >= N ) then
        //print( Unquoted, "Invalid value for a: ", a );
        return( FALSE );
    end_if;

    if( b >= N ) then
        //print( Unquoted, "Invalid value for b: ", b );
        return( FALSE );
    end_if;

    if( c >= N ) then
        //print( Unquoted, "Invalid value for c: ", c );
        return( FALSE );
    end_if;

    if( x >= N ) then
        //print( Unquoted, "Invalid value for x: ", x );
        return( FALSE );
    end_if;

    if( r >= N ) then
        //print( Unquoted, "Invalid value for r: ", r );
        return( FALSE );
    end_if;

    if( R >= N ) then
        //print( Unquoted, "Invalid value for R: ", R );
        return( FALSE );
    end_if;

    // Compute A, B, C
    A := (a * powermod( g1, f1(a), N )) mod N;
    //print(A);
    B := (b * powermod( g2, f2(powermod(h2, b, t)), N )) mod N;
    //print(B);
    C := (c * powermod( g3, f3(powermod(h3, c, t)), N )) mod N;
    //print(C);

    // Verify the signature
    return( powermod( R, v, N ) = (powermod( A, x, N ) * B * powermod( C, r, N )) mod N );
end_proc;
```

```

// find prime p in the interval [a,b] s.t. P | p-1
bank::randomprimePQ := proc(a, b, P)
    local p1, pi2, rg;
begin
    SEED:=rtime();
    rg := 1 + P*random((a+P-1) div P..b div P);
    cryptotools::randomprimefrom(rg)
end_proc;

// find g s.t. it has order P*Q in Z_N*
bank::generate_g := proc( p,q,P,Q )
    local N, L, rnd_gen, x, modN;
begin
    N := p*q;
    L := (p-1)*(q-1);
    modN := Dom::IntegerMod(N);
    rnd_gen := modN::random;
    repeat
        repeat
            x := rnd_gen();
        until igcd(expr(x),N)=1 end_repeat;
        x := x^(L/(P*Q));
    until (expr(x) <> 1) and (expr(x^P) <> 1) and
        (expr(x^Q) <> 1) end_repeat;
    return( x );
end_proc;

// find h s.t. is has order N in F_t*
bank::generate_h := proc( p,q,t )
    local N, rnd_gen, x, modt;
begin
    N := p*q;
    modt := Dom::IntegerMod(expr(t));
    rnd_gen := modt::random;
    repeat
        repeat
            x := rnd_gen();
        until igcd(expr(x),t)=1 end_repeat;
        x := x^((t-1)/N);
    until (expr(x) <> 1) and (expr(x^expr(p)) <> 1) and
        (expr(x^expr(q)) <> 1) end_repeat;
    return( x );
end_proc;

// generate RSA keys, s.t. size of the public key is sizeOfv
bank::generatekey := proc( p,q,sizeOfv )
    local N, L, v, v_inv, rnd_gen;
begin
    N := p*q;
    L := (p-1)*(q-1);

    rnd_gen := random((2^(sizeOfv-1))..(2^sizeOfv));
    repeat
        v := rnd_gen();
    until igcd(v,L)=1 end_repeat;
    v_inv:=1/v mod L;

    return( [[N,v], [N,v_inv]] );
end_proc;

// hash function(s), Z_N* -> N < v
f1 := proc(i)
    local ret;
begin
    ret := SHA512(numlib::g_adic(i,2^64));
    return( _plus(op(zip( ret, [2^(64*(i-1))$i=1..nops(ret)], _mult ))) mod v);
// return((ret[1] + ret[2]*(2^64) + ret[3]*(2^(2*64)) + ret[4]*(2^(3*64))
// + ret[5]*(2^(4*64)) + ret[6]*(2^(5*64)) + ret[7]*(2^(6*64))
// + ret[8]*(2^(7*64))) mod v);
end_proc;
f2 := f1;
f3 := f1;
f4 := proc(i, j)
    local ret;
begin
    ret := SHA512(numlib::g_adic(i,2^64).numlib::g_adic(j,2^64));
    return( _plus(op(zip( ret, [2^(64*(i-1))$i=1..nops(ret)], _mult ))) mod v);
end_proc;

```

```
//Calculation [tm]
bank::setup := proc( x = 1024 )
begin
    sizeofP := x div 8:
    sizeofQ := x div 8:
    sizeofp := x div 2:
    sizeofq := x div 2:
    sizeofv := x div 8:
    sizeoft := sizeofp + sizeofq + 32:
    P := randomprime(2^(sizeofP-1), 2^sizeofP);
    Q := randomprime(2^(sizeofQ-1), 2^sizeofQ);
    print(Unquoted, NoNL, "Generating RSA Keys...");
    p := bank::randomprimePQ(2^(sizeofp-1), 2^sizeofp, P);
    q := bank::randomprimePQ(2^(sizeofq-1), 2^sizeofq, Q);
    [public, secret] := bank::generatekey(p, q, sizeofv);
    print(Unquoted, "done");
    v := public[2];
    N := public[1];
    v_inv := secret[2];
    print(Unquoted, NoNL, "Generating g's...");
    g1 := bank::generate_g(p, q, P, Q);
    g2 := bank::generate_g(p, q, P, Q);
    g3 := bank::generate_g(p, q, P, Q);
    print(Unquoted, "done");
    print(Unquoted, NoNL, "Generating t...");
    t := bank::randomprimePQ(2^(sizeoft-1), 2^sizeoft, (p*q));
    print(Unquoted, "done");
    print(Unquoted, NoNL, "Generating h's...");
    h2 := bank::generate_h(p, q, t);
    h3 := bank::generate_h(p, q, t);
    print(Unquoted, "done");

    print(Unquoted, NoNL, "Writing to files...");
    io::open("X:\\io\\bank_private.mu"):
    io::wipe():
    io::send(hold(N), hold(v_inv));

/**/
    io::open("X:\\io\\bank_public.mu"):
    io::wipe():
    io::send(hold(N), hold(v), hold(t),
        hold(g1), hold(g2), hold(g3),
        hold(h2), hold(h3),
        hold(f1), hold(f2), hold(f3), hold(f4)
    ):
    print(Unquoted, "done");
    io::open("X:\\io\\alice_id.mu"):
    U:=random(100..999)():
    io::send(hold(U)):
    print(Unquoted, "Bank-Setup complete!");
end_proc;
```



```

// SHA512 acc.to FIPS180-2
// (c) 2006 MNüsken

TypeWordlist := ()->Type::ListOf( Type::Interval( [0,2^64-1],Type::Integer ), args() ):

text2wordlist := proc(t)
    local k,i;
begin
    numlib::toAscii(t);
    [ _plus( %[8*k+i]*256^(8-i) $ i=1..min(8,nops(%)-8*k) ) $ k=0..((nops(%)+7) div 8)-1 ], leng
th(t)*8;
end_proc:
protect(text2wordlist):

ROTL := proc(w,l,x)
begin
    if x = 2^w-1 then return(x); end_if;
    x := modp(x * 2^modp(1,w), 2^w-1);
end_proc:
protect(ROTL):

ROTR := (w,l,x)->ROTL(w,w-l,x):
protect(ROTR):

SHR := (w,l,x)-> x div 2^l:
protect(SHR):

bitlist := (x,w)->numlib::q_adic(x+2^w,2)[1..w]:
protect(bitlist):

revbitlist := proc(x,w) local y, i; begin y:=bitlist(args()); [y[i]$i=w..1 step -1]; end_proc:
protect(revbitlist):

bitwise_xor := proc()
    local n, x, i, j;
begin
    n := max( op(map( [args()], x->1+floor(log(2,1+abs(x))) )) );
    x := map([args()],bitlist,n);
    _plus( 2^(i-1) * ( _plus(x[j][i] $ j=1..nops(x)) mod 2) $ i=1..n );
end_proc:
protect(bitwise_xor):

cryptotools::SHA512
:=proc( M: TypeWordlist(), l )
    local Ch, H, K, L, Maj, SHA512_compress, Sigma0, Sigma1, i, k, lw, revbitlist, sigma0, sigma
1, w;
    save DIGITS;
begin
    dprint2( "SHA512 on ",M, "... " );
    w := 64;
    Ch := proc(x,y,z)
        local i;
        begin
            [x,y,z] := map([x,y,z],bitlist,w);
            _plus( 2^(i-1) * _if(x[i]=1,y[i],z[i]) $ i=1..nops(x) );
        end_proc;
    Maj := proc(x,y,z)
        local i;
        begin
            [x,y,z] := map([x,y,z],bitlist,w);
            _plus( 2^(i-1) * ((x[i]*y[i]+y[i]*z[i]+z[i]*x[i])mod 2) $ i=1..nops(x) );
        end_proc;
    Sigma0 := x->bitwise_xor( ROTR(w,28,x), ROTR(w,34,x), ROTR(w,39,x) );
    Sigma1 := x->bitwise_xor( ROTR(w,14,x), ROTR(w,18,x), ROTR(w,41,x) );
    sigma0 := x->bitwise_xor( ROTR(w,1,x), ROTR(w, 8,x), SHR (w, 7,x) );
    sigma1 := x->bitwise_xor( ROTR(w,19,x), ROTR(w,61,x), SHR (w, 6,x) );
    DIGITS:=ceil(w/3):
    K := [ floor(frac(ithprime(i)^(1/3))*2^w) $ i=1..80 ];
    userinfo( 20, "constant K hex = ",map( K, int2text, 16 ) );
    // Padding (assuming M consists of entire words):
    if args(0)<3 then l := nops(M)*w; else
        if nops(M)>(l+w-1) div w or M[nops(M)] mod 2^modp(-1,w)<>0 then
            error("Message contains dirt after claimed length");
        end_if;
    end_if;
    lw := l div w;
    k := modp(-2*w-1-1,16*w);
    L := numlib::q_adic( 2^(2*w+k)+1, 2^w );

```

```

M := M . [ L[i] $ i=nops(L)..1 step -1 ];
if nops(M) mod 16<>0 then M:=[ op(M,1..lw), M[lw+1]+M[lw+2], op(M,lw+3..nops(M)) ]; end_if;
userinfo( 2, "Padded message words = ", map( M, int2text, 16 ) );
userinfo( 10, "Padded message bits = ", map( M, op@revbitlist, w ) );
SHA512_compress := proc( H, W )
    local t, T, T1, T2;
    begin
        W := W . [0$64];
        for t from 17 to 80 do
            W[t] := modp( sigma1(W[t-2]) + W[t-7] + sigma0(W[t-15]) + W[t-16], 2^w );
        end_for;
        T := H;
        for t from 1 to 80 do
            T1 := modp( T[8] + Sigma1(T[5]) + Ch(T[5],T[6],T[7])
                + K[t] + W[t], 2^w );
            T2 := modp( Sigma0(T[1]) + Maj(T[1],T[2],T[3]), 2^w );
            T := [ modp(T1+T2,2^w), T[1], T[2], T[3],
                modp(T[4]+T1,2^w), T[5], T[6], T[7]];
            userinfo( 3, "t=".(t-1)." abcdefg = ",map( T, int2text, 16 ) );
        end_for;
        [ modp(T[i]+H[i],2^w) $ i=1..8 ];
    end_proc;
H := [ floor(frac(ithprime(i)^(1/2))*2^w) $ i=1..8 ];
userinfo( 20, "initial H hex = ",map( H, int2text, 16 ) );
for i from 1 to nops(M) step 16 do
    H := SHA512_compress( H, M[i..i+15] );
    userinfo( 2, "after round ".i." H hex = ",map( H, int2text, 16 ) );
end_for;
dprint2( "SHA512 yields ", H );
H;
end_proc;

```

```
// from Bank
```

```
N := 60437657139153073782634106064054444710352822949122744799276571266651030210783629176230164
3195007192702793694793636443997114659447287412738477371068090782730159000248062093179472947693
6213942653868526446943593698484719038826968783793737395000193588132899834801377832265375542582
0321492752672253726296558811013:
v := 258467192543354217231020638930256186123:
t := 65104822671942063742807219588256450731712569897628021340221359669197368253102142690865755
2863238096508493340647502335940070153350656015285910150567949341103409001803531144325087403495
0870143258137283468130314574549220689611529300579665905662976302551924249094576515940483010889
36469960654637619100573887968400063734911:
g1 := 3010074410931292675498187393052958101751325584718134112185679307135919484687945000092758
0857591904903114680615569900517923610840091388341601840350842577666494486115802151802390570932
7523211450633095209374401549801454881680164006794825841334318911257838873183384170418532440655
1662121934142038782852921432808 mod 6043765713915307378263410606405444471035282294912274479927
6571266651030210783629176230164319500719270279369479363644399711465944728741273847737106809078
2730159000248062093179472947693621394265386852644694359369848471903882696878379373739500019358
81328998348013778322653755425820321492752672253726296558811013:
g2 := 5946207306609701364311804882015695095142888753180693974946482884717989164563630868749098
1508539599218387603430425678855338798149687210880164025825408966917136951302925207867052513680
9627253132910933661422390364547081435411049641677533570466996082871242637660964972084037177677
122709752906165439185538061476 mod 6043765713915307378263410606405444471035282294912274479927
6571266651030210783629176230164319500719270279369479363644399711465944728741273847737106809078
2730159000248062093179472947693621394265386852644694359369848471903882696878379373739500019358
81328998348013778322653755425820321492752672253726296558811013:
g3 := 1656637477346231423219089506281257395366787610479753288125057040127874490159429604914092
7147151854214839370419055671982691944252436095278386585835329851477676817865957462681088172028
9662188405925532274919198131182683327013978270580785201122949465017285824600976619632521550789
93984056059806539638936342912287 mod 6043765713915307378263410606405444471035282294912274479927
76571266651030210783629176230164319500719270279369479363644399711465944728741273847737106809078
82730159000248062093179472947693621394265386852644694359369848471903882696878379373739500019358
881328998348013778322653755425820321492752672253726296558811013:
h2 := 4749425678335803966023053867325622065253938269864060176376381060510910016373629488646458
5489290072210768703441060339829807782125884265438971993356697466777385099546187795502715440573
5891863882918615194856618120735024393897666037430289186405405522764792674054145335774907619254
604484838757285514766367393140644406821079 mod 65104822671942063742807219588256450731712569897
6280213402213596691973682531021426908657552863238096508493340647502335940070153350656015285910
1505679493411034090018035311443250874034950870143258137283468130314574549220689611529300579665
90566297630255192424909457651594048301088936469960654637619100573887968400063734911:
h3 := 9645182379957654110422660921492044343438585890913662147419578158476617037871205580832597
6392994009054372419983310820813072284355990695222073465159847996841057417705084144437669627991
8626836285415048755986742632079260669210475161555082470532255818674204879873637269451085444369
74992923315965661708541170482109176277399 mod 651048226719420637428072195882564507317125698976
2802134022135966919736825310214269086575528632380965084933406475023359400701533506560152859101
5056794934110340900180353114432508740349508701432581372834681303145745492206896115293005796659
0566297630255192424909457651594048301088936469960654637619100573887968400063734911:
f1 := proc(i)
  name f1;
  local ret;
begin
  ret := SHA512(numlib::g_adic(i, 2^64));
  return(_plus(op(zip(ret, [2^(64*(i - 1)) $ i = 1..nops(ret)], _mult))) mod v)
end_proc:
f2 := proc(i)
  name f1;
  local ret;
begin
  ret := SHA512(numlib::g_adic(i, 2^64));
  return(_plus(op(zip(ret, [2^(64*(i - 1)) $ i = 1..nops(ret)], _mult))) mod v)
end_proc:
f3 := proc(i)
  name f1;
  local ret;
begin
  ret := SHA512(numlib::g_adic(i, 2^64));
  return(_plus(op(zip(ret, [2^(64*(i - 1)) $ i = 1..nops(ret)], _mult))) mod v)
end_proc:
f4 := proc(i, j)
  name f4;
  local ret;
begin
  ret := SHA512(numlib::g_adic(i, 2^64).numlib::g_adic(j, 2^64));
  return(_plus(op(zip(ret, [2^(64*(i - 1)) $ i = 1..nops(ret)], _mult))) mod v)
end_proc:
```

```
// from Bank
```

```
N := 60437657139153073782634106064054444710352822949122744799276571266651030210783629176230164
3195007192702793694793636443997114659447287412738477371068090782730159000248062093179472947693
6213942653868526446943593698484719038826968783793737395000193588132899834801377832265375542582
0321492752672253726296558811013:
```

```
v_inv := 366957533673330799390676354255549005865596601238597925212251096351799695097024419608
8358056672348183847636700328043583138575380040470079410703042221622539896213619431946251792672
8304620915334959720816406982296673276550661183930946703716799803150659452026122449479781361357
98526894706174374590706912774281367:
```

```
// from Bank  
U := 515:
```



```
// requires global setup: N,h2,h3,t
bank::withdraw:=proc()
begin
  SEED:=rttime();

  //creates a random invertible element, argument is a randomNumberGenerator
  createRandomInvertible:=proc(randN)
    local result;
  begin
    repeat
      result:=randN();
    until igcd(result,N)=1 end_repeat:
    return (result):
  end_proc:

  // Withdrawal step 2

  print(Unquoted, NoNL, "Receiving Global Setup..."):
  io::open("X:\\io\\bank_public.mu", FALSE):
  delete N:
  io::waitfor( N ):
  io::open("X:\\io\\bank_private.mu", FALSE):
  io::receive():
  print(Unquoted, " Done!"):

  print(Unquoted, NoNL, "Receiving Step 1 from Alice..."):
  io::open("X:\\io\\withdraw.mu", FALSE):
  io::whoami:="Bank":

  a_pp:=FAIL:
  io::waitforwipe(A_t,a_pp):
  print(Unquoted, " Done!"):
  print(Unquoted, NoNL, "Executing Step 2..."):

  a_pp:=createRandomInvertible(random(1..N)):
  b_pp:=createRandomInvertible(random(1..N)):
  c_pp:=createRandomInvertible(random(1..N)):

  h2_t:=powermod(h2,b_pp,t);
  h3_t:=powermod(h3,c_pp,t);

  io::send( hold(a_pp), hold(h2_t), hold(h3_t) ):
  print(Unquoted, " Done!"):

  //Withdrawal step 4
  print(Unquoted, NoNL, "Receiving Step 3 from Alice..."):
  io::waitfor(e1):
  print(Unquoted, " Done!"):
  /*
  print(Unquoted, NoNL, "Receiving private key from Bank..."):
  io::open("X:\\io\\bank_private.mu", FALSE):
  io::receive():
  print(Unquoted, " Done!"):
  */
  print(Unquoted, NoNL, "Verifying Signature (ie. Executing Step 4)..."):

  A_b:=A_t * a_pp * f4(e2,e3)*powermod(g1,e1,N) mod N:
  B_b:= B_t * b_pp * powermod(g2,e2, N) mod N:
  C_b:= C_t * c_pp * powermod(g3,e3, N) mod N:

  k_pp:= createRandomInvertible(random(1..v)):

  // compute the signature
  S1_t:= rsa::encrypt([N,v_inv], A_b*powermod(C_b,k_pp,N) mod N ):
  io::open("X:\\io\\alice_id.mu", FALSE):
  io::receive():
  // compute the signature
  S2_t:= rsa::encrypt([N,v_inv], B_b*powermod(C_b,U,N) mod N ):
  io::open("X:\\io\\withdraw.mu", FALSE):
  io::whoami:="Bank":
  io::send( hold(S1_t), hold(S2_t), hold(b_pp), hold(c_pp), hold(k_pp) ):
  print(Unquoted, " Done!"):
end_proc:
```

```

customer::withdraw:=proc()
begin
  SEED:=rtime();

  io::open("X:\\io\\bank_public.mu", FALSE):
  delete N:
  io::waitfor(N);

  io::open("X:\\io\\alice_id.mu", FALSE):
  delete U:
  io::waitfor(U):

  io::open("X:\\io\\withdraw.mu", FALSE):
  io::whoami:="Alice";
  io::wipe():

  //proc for generating random which is invertible
  createRandomInvertible:= proc(randN)
    local res;
  begin
    repeat
      res:=randN():
    until igcdex(res, N)[1]=1
    end_repeat:
    return(res):
  end_proc :

  //generate randoms
  randN:=random(1..N):
  a_p:=createRandomInvertible(randN):
  b_p:=createRandomInvertible(randN):
  c_p:=createRandomInvertible(randN):
  alpha:=createRandomInvertible(randN):
  unprotect(beta):
  beta:=createRandomInvertible(randN):
  unprotect(gamma):
  gamma:=createRandomInvertible(randN):
  randV:=random(1..v):
  sigma:=createRandomInvertible(randV):
  tau:=createRandomInvertible(randV):
  phi:=createRandomInvertible(randV):

  //compute
  A_t:=powermod(alpha, v, N) * a_p * powermod(g1, sigma, N) mod N:
  B_t:=powermod(beta, v, N) * b_p * powermod(g2, tau, N) mod N:
  C_t:=powermod(gamma, v, N) * c_p * powermod(g3, phi, N) mod N:

  //send them
  io::send(hold(A_t), hold(B_t), hold(C_t));

  //receive step 2
  io::waitfor(a_pp);

  //Step 3:
  withdrawal3:=proc(a_pp, h_2t, h_3t)
    local e_1, e_2, e_3, k_m;
  begin
    e_2:=f2(powermod(h_2t, b_p, t))-tau;
    e2_h:=e_2 div v;
    e_2:=e_2 mod v;
    e_3:=f3(powermod(h_3t, c_p, t))-phi;
    e3_h:=e_3 div v;
    e_3 := e_3 mod v;
    k_p := createRandomInvertible(random(1..v));
    k_m := igcdex(k_p, v)[2];
    one_h:=k_p*k_m div v;
    a:=powermod(a_p*a_pp*f4(e_2, e_3), k_p, N);
    e_1:=k_m*f1(a) - sigma;
    e1_h:=e_1 div v;
    e_1:=e_1 mod v;
    return ([e_1, e_2, e_3]);
  end_proc:

  [e1, e2, e3]:=withdrawal3(a_pp, h2_t, h3_t):
  io::send(hold(e1), hold(e2), hold(e3));

```

```

//step 5:
io::waitfor(b_pp):

b:=b_pp*b_p mod N:
c:=c_pp*c_p mod N:
k:=k_p*k_pp:
k_h:=k div v:
k:=k mod v:
A:=a*powermod(g1, f1(a),N) mod N:
B:=b*powermod(g2, f2(powermod(h2,b,t)),N) mod N:
C:=c*powermod(g3, f3(powermod(h3,c,t)),N) mod N:
S1End:=powermod(q1,-1*f1(a)*one_h,N)*powermod(C,-1*k_h,N) mod N:

S1:=((powermod( S1_t
      *((1/alpha mod N)*powermod(g1, e1_h,N))
      *powermod(igcdex(gamma, N)[2]*powermod(q3, e3_h, N), k_pp,N),
      k_p,N))) *S1End mod N:
S2_mid:=igcdex(beta, N)[2]*powermod(g2, e2_h,N) mod N:
S2_end:=powermod((igcdex(gamma, N)[2]*powermod(g3, e3_h,N) mod N), U,N) mod N:
S2:=S2_t*S2_mid*S2_end mod N:

//check signatures
if powermod(S1,v,N)=(A*powermod(C,k,N) mod N) and powermod(S2,v,N)=(B*powermod(C,U,N) mod N)
then
  io::open("X:\\io\\coin.mu", FALSE):
  io::wipe():
  io::send(hold(a),hold(b),hold(c),hold(k),hold(S1),hold(S2)):
  print("Coin stored!"):
  //return ([a,b,c,k,S1,S2]):
else
  print("Error");
end_if:
end_proc:

```

```
// from Alice
A_t := 482412716378275206426700898279616078506522042606538925322193393071987166014048724618445
6481558270294853564948836466568227395616179063975107067773293128780907360154796537009402225858
1361819053682067055245556955701427047542729221564737169548586365084733134868512689536735400074
207799313979581149457517005421980:
B_t := 562677960811918916806528491286822610646974995449016007156259285821213473404141762075671
1042772028941425862722511416258731160305444945022934818764957038458282068115707850320757210655
546597488668769180852329317433116921603591695407451053410236537077553218421660470915735377970
486266667144841293206365840429808:
C_t := 349044521169362681059004976271198006760685153007192324177988622383943210094638658250987
5445339712843567240176884502120830250892946326548709033903640596136670664871969387613943270577
1997320379596905948075817176722789707451070316892318044124825754018019125286353642270265996852
098548512939949327714328252741140:
// from Bank
a_pp := 55614131527750711471482819392784931667155976378931383636070158419812838181988417056447
5870166763939485271941783715379948613053180669310312671233411999249513161433374773472851364064
5888069297442692526467055793444879840224689395229057946199272361888586709914055322695966312606
4922706232368345877384338830586830:
h2_t := 13100184329317927234259586073927705511545352129469511949428637268116112880554898001403
0715867215134728748633356286052128139169590748120503205148731461410923886795068956657879804127
6388437688907919009179493747906576080688412016294550812075929073818804183682112022574193257918
95620503525754996463078299446220201780257124:
h3_t := 36892300600117999257079261650724371179145967508812401587454693513566006635173408493816
1243151592631555364006820307225979274163602370933967125956105976653705487302950424109383058889
6749895546291749534872768903414198532520354345238456274482285950486400336434903422365645435596
08413785176963251782733496827445124347268947:
// from Alice
e1 := 153623389446834055656834587072432574089:
e2 := 50535717683996291760072864483947135654:
e3 := 41403809454054333619753336047200267460:
// from Bank
s1_t := 17587416892859489446509723834877092138276358010745928095410461212896985723356086867155
7720311484007884514230189431558809804738417726680613095809238553999018222639729152436102553924
275266388165227228885844482045518953590787522586378063504934924082238874558171819584637011804
5582706359670465949372300281974210:
s2_t := 11087564082714181075544199897805260818435944354629558753064375569729307668946570005129
5159503469573656173491142365049419458912628691944634660600487029759202991303917596098803442630
7479269631363837662972823851582536300724847759157273856853792450323314214293844523055371783735
6937971362466344601357352859165914:
b_pp := 49281738557972824310126056933882344048630929022487344876957702080149602739346488439333
3304837379458009532962550054794614029817234093957754820208195589998102995922863780080480880857
4875543640300085687897981545818492269367591359677311281793298883309091843463641810989701465879
4842897499330271254267293512614726:
c_pp := 15033837812431143537679699142199507069842786260857123287019325538115721898757110836404
1541268459549763452055976218898465026200844581028186440117177620363958990375395272401260528471
4686957304908856877836937661957495997170434120900669868661563983933794632243272867367980876322
9687592687512195723824467627322227:
k_pp := 87511358334039923421892284838550949679:
```

```
customer::payment := proc()
  local handover, response2challenge;
begin
  io::open("X:\\io\\bank_public.mu", FALSE):
  delete N:
  io::waitfor( N ):

  io::open("X:\\io\\alice_id.mu", FALSE):
  delete U:
  io::waitfor( U ):

  handover := proc()
    save a,b,c;
  begin
    io::open("X:\\io\\coin.mu", FALSE):
    io::receive():
    io::open("X:\\io\\payment.mu", FALSE):
    io::wipe():
    io::send(hold(a),hold(b),hold(c)):
  end_proc:

  response2challenge := proc()
    save r, R;
    local temp, r_h, C;
  begin
    io::waitfor( x );
    temp:=k*x+U;
    r:=temp mod v;
    r_h:=temp div v;
    C := (c * powermod( g3, f3(powermod(h3, c, t)), N )) mod N;
    R :=(powermod(S1,x,N)*S2 * powermod( C, (-r_h),N)) mod N;
    io::send(hold(r),hold(R)):
  end_proc:

  handover(a,b,c);
  //print(x):
  response2challenge();
end_proc:
```



```
/* Generates the random challenge*/
merchant::challenge := proc ()
save x, v;
begin
  //print(v):
  x:= random(0 .. v-1());
  io::send(hold(x));
end_proc:

/* Verify that Signature from Alice is valid + check if all
transmitted data are in the required domains */

merchant::verifySig := proc ()
save v, R, r;
begin
  io::receive();
  return( bool(
    ferguson::is_valid_signature(a, b, c, x, r, R) and
    (r < v) and (R < N)
  )) //end_of return
end_proc:

/*Procedure to start Martin in Payment protocol 2*/
merchant::payment:= proc ()
begin
  print(Unquoted,NoNL, "Receiving Bank public"):
  io::open("X:\\io\\bank_public.mu", FALSE):
  delete N:
  io::waitfor(N):
  print(Unquoted,"... Done"):
  print(Unquoted,"Initialising"):
  io::open("X:\\io\\payment.mu", FALSE):
  print(Unquoted,"Waiting for variables!"):
  x:=FALL:
  io::waitforwipe(c,x);
  print(Unquoted,"Variables received! Start generating challenge."):
  merchant::challenge();
  print(Unquoted,"Challenge sent!"):
  io::waitfor(r, R);
  print(Unquoted,"verifying signature...");
  if (merchant::verifySig()) then
    print("Coin verified !!")
  else
    print("Goto jail, Alice")
  end_if;
end_proc:
```

```
// from Alice
a := 58845901278554930641312538065507857913714654825308149381019957975069031556837337222638069
5758146934506889799291613772573419368310952044235836630132100382873168305022524271152186811104
0899306146556455524738359921976278401121707042139643985759251007072439379585036666078594347577
7787284022132898600612544864213:
b := 19528459919923323887790935199316675380438890861975906798868271047060693857091434185278936
1824219533938874968839234026743012545944278785924657018382987373178594336494607469476349265162
6899277544675185656720492159152797905494862688619676509210856367796797606400692836780129888355
5770983155344010496489917953313:
c := 13811548696764119782577406315965372856842452482617842350858430080099650455787259645738179
4101027341869531546895577684853273561126962911621101443506491246996974499033691446490503250734
8481448091089143510992139221201229365328124509147813183175511290120473897061794174928115998662
6647831234152831750080887479060:
// from Martin
x := 246173918657395679812888740849718982915:
// from Alice
r := 255780981169105824414814852286228235004:
R := 15096273079079492012532378765295517357229180018067428143273969670667944979182512807986593
9823692937700389529949865240962197891008639611813149942152515747211327566564219562805867868146
2553188536327840720200232560106901391501116709272148544998207934400940674106914935238039248880
3865491148729775738401357469235:
```

```
/* Send entire transcript of Payment protocol 2 to the bank */
merchant::deposit:= proc()
save a,b,c,x,r,R;
begin
  print("deposit...");
  io::open("X:\\io\\deposit.mu",FALSE):
  io::wipe();
  io::send(hold(a),hold(b),hold(c),hold(x),hold(r),hold(R));
  print("...done");
end_proc;
```

```
database := [];          /* global storage for registered coins */

//-----

/* Store a coin in the internal database */
register_coin := proc( a, b, c, x, r, R )
begin
    database := database . [[a, b, c, x, r, R]];
    return( );
end_proc;

//-----

/* Returns TRUE/FALSE if the given coin is already present in the database */
check_no_double_spending := proc( a, b, c, x, r ) : DOM_BOOL
begin
    // initially, no double spending is assumed
    d := FALSE;

    // See if x, r already exist in the database of deposited coins
    i := 1;
    while i <= nops(database) do

        if database[i][1] = a and database[i][2] = b and database[i][3] = c then
            print( Unquoted, "*** DOUBLE SPENDING ALERT ***" );
            d := TRUE;

            if database[i][4] = x then
                print( Unquoted, "The merchant cheated!" );
            else
                // Figure out the client's identity
                u:=(database[i][5]*x - r*database[i][4])/(x-database[i][4]) mod v;
            //    u:=(database[i][5]*x - r*database[i][4])*igcdex((x-database[i][4]) mod v, v)[2] mod v;
                print( Unquoted, "The client " . u . " cheated!" );
            end_if;

            break;
        end_if;

        i := i + 1;
    end_while;

    return( d );
end_proc;

//-----

showimage := proc( success : DOM_BOOL )
begin
    if success then
        f := fopen( "X:\\CRYPTO\\lib\\FERGUSON\\success.txt" );
    else
        f := fopen( "X:\\CRYPTO\\lib\\FERGUSON\\failure.txt" );
    end_if;

    linecounter := 1;
    image := "";

    repeat
        ftextinput( f, x );
        print( Unquoted, x );
        linecounter := linecounter + 1;
    until linecounter = 36 end_repeat;

    null();
end_proc;

//-----

bank::deposit := proc()
begin
    print(Unquoted, NoNL, "Receiving Global Setup...");
    io::open("X:\\io\\bank_public.mu", FALSE);
    delete N;
    io::waitfor( N );
    io::open("X:\\io\\bank_private.mu", FALSE);
    io::receive();
    print(Unquoted, " Done!");
end;
```

```
io::open("X:\\io\\deposit.mu"):

print( Unquoted, "===== " );
print( Unquoted, "Waiting for coin..." );
receipt:="missing";
io::waitforwipe(r,receipt);
receipt:="Thanks";
io::send(hold(receipt));

print( Unquoted, "===== " );
print( Unquoted, "a: " . a );
print( Unquoted, "b: " . b );
print( Unquoted, "c: " . c );
print( Unquoted, "x: " . x );
print( Unquoted, "r: " . r );
print( Unquoted, "R: " . R );
print( Unquoted, "===== " );

print( Unquoted, NoNL, "Checking signature... " );

if not ferguson::is_valid_signature(a, b, c, x, r, R) then
  print( Unquoted, "Signature is invalid - aborting!" );
  showimage( FALSE );
  return();
else
  print( Unquoted, "ok" );
end_if;

if not check_no_double_spending( a, b, c, x, r ) then
  print( Unquoted, "Crediting the merchant a dollar..." );

  print( Unquoted, "Registering coin..." );
  register_coin( a, b, c, x, r, R ):

  showimage( TRUE );
else
  showimage( FALSE );
end_if;

print( Unquoted, "Done." );

end_proc:

//-----
```

```
// from Martin
```

```
a := 58845901278554930641312538065507857913714654825308149381019957975069031556837337222638069
5758146934506889799291613772573419368310952044235836630132100382873168305022524271152186811104
0899306146556455524738359921976278401121707042139643985759251007072439379585036666078594347577
7787284022132898600612544864213:
```

```
b := 19528459919923323887790935199316675380438890861975906798868271047060693857091434185278936
1824219533938874968839234026743012545944278785924657018382987373178594336494607469476349265162
6899277544675185656720492159152797905494862688619676509210856367796797606400692836780129888355
5770983155344010496489917953313:
```

```
c := 13811548696764119782577406315965372856842452482617842350858430080099650455787259645738179
4101027341869531546895577684853273561126962911621101443506491246996974499033691446490503250734
8481448091089143510992139221201229365328124509147813183175511290120473897061794174928115998662
6647831234152831750080887479060:
```

```
x := 246173918657395679812888740849718982915:
```

```
r := 255780981169105824414814852286228235004:
```

```
R := 15096273079079492012532378765295517357229180018067428143273969670667944979182512807986593
9823692937700389529949865240962197891008639611813149942152515747211327566564219562805867868146
2553188536327840720200232560106901391501116709272148544998207934400940674106914935238039248880
3865491148729775738401357469235:
```

```
// from Bank
```

```
receipt := "Thanks":
```


Plan

e€
4.04.08
⑦

- Security of e€
and Ferguson (1994)
- Polish (your) implementation
and demonstrate it
- Google for new developments
- Final discussion, add ons,
more features?, ...

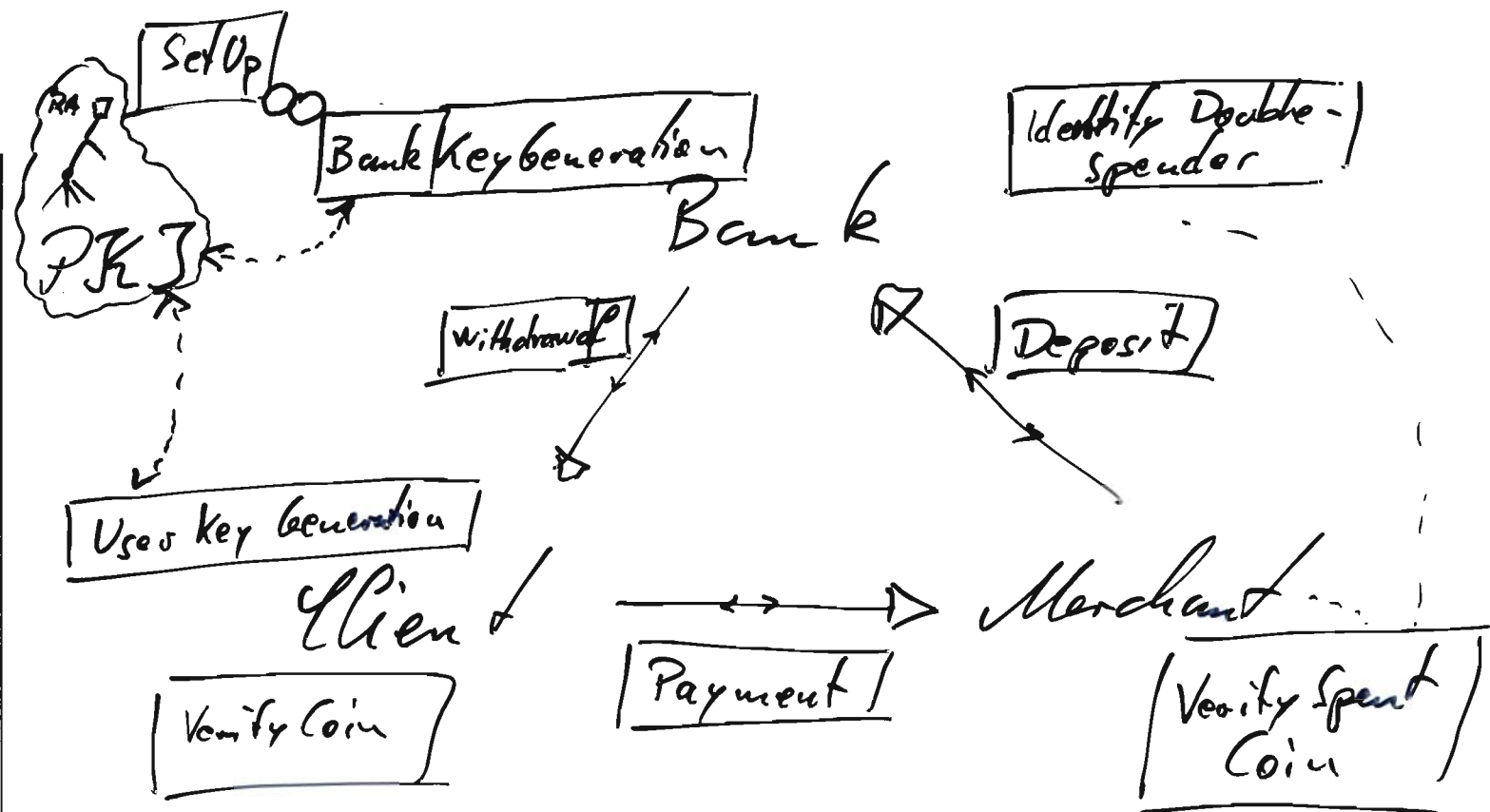
Security of $e\epsilon$

based Trolin (2006?)
and Ferguson (1994)

$e\epsilon$
4.0408
(2)

We want to formulate security goals
like with encryption and signatures.

Parts of a (single) anonymous $e\epsilon$ system



Experiment CORRECTNESS

Setup

Bank Key Generation

User Key Generation

if \rightarrow Verify Coin then error

\rightarrow adversary defines new coin and transaction id

Payment

Accept \ if \rightarrow Verify Spent then error

Experiment / UNFORGEABILITY

e€
4.04.02
(3)

Setup

Bank Key Generation

Adversary may query honest Withdrawals
and output a list of ~~Withdrawals~~ k spent coins

if $k \leq \# \text{queries}$
then Return (Adv fail)

if $\exists i : \text{Verify Coin } i$
then Return (Adv fail)

if $\exists i \neq j : \text{Identify Double Spender } (\text{coin}_i, \text{coin}_j)$
 \in one corrupt client

then Return (Adv fail)

Return (Adv succeeded !)

In our example:

e €
9.09.08
(4)

the adversary gets coins

$$(a^{(i)}, b^{(i)}, c^{(i)}, k^{(i)}, S_1^{(i)}, S_2^{(i)})$$

with

$$S_1^v = A C^k, \quad S_2^v = B C^u$$

and shall construct k deposit spent coins

$$(a^{(i)}, b^{(i)}, c^{(i)}, x^{(i)}, r^{(i)}, R^{(i)})$$

with

$$R^v = A^x B C^r$$

As one simple attack we can ask
whether multiplying two coins gives
a new one:

$$\begin{aligned} \text{say } S_1 &= S_1^{(1)} S_1^{(2)} \\ S_2 &= S_2^{(1)} S_2^{(2)} \\ A &= A^{(1)} A^{(2)} \\ B &= B^{(1)} B^{(2)} \\ C^k &= (C^{(1)k^{(1)}} C^{(2)k^{(2)}}) \end{aligned} \Rightarrow \begin{aligned} S_1^v &= A C^{(1)k^{(1)}} C^{(2)k^{(2)}} \\ S_2^v &= B C^u \end{aligned}$$

find C ? \rightarrow break RSA! or (#)

further how to find a such that

$$\begin{aligned} A &= a g_x^{f_1(a)} \quad \text{given } a^{(1)}, a^{(2)} \text{ with} \\ &= a^{(1)} a^{(2)} g_x^{\frac{f_1(a^{(1)}) + f_1(a^{(2)})}{2} - \frac{f_1(a^{(1)} a^{(2)})}{2}} \quad A^{(i)} = a^{(i)} g_x^{f_1(a^{(i)})} \end{aligned}$$

Thus f_1 should be one-way & coll.-resist. e€
4.04.02
(5)
or actually $a \mapsto a_{g_1}^{f_1(a)}$

must be one-way
and collision-resistant.

Also $b \mapsto b_{g_2}^{f_2(h_2^b)}$

must be one-way & collision-resistant.

(#) Further observation: if the adversary would be able to force the same k in two withdrawals then he could solve the "C-problem" seen above.

... continue along these lines.

Do not forget that you are a designer!

Experiment NONFRAMABILITY

e€
4.04.08

(6)

Adversary executes Bank Setup
and BankKey Generation

Adversary with access to
oracle for UserKey Generation,
oracle for User - Withdrawal,
oracle for Payment
shall produce a list of spent coins.
and a user id j .

if $\exists i : \rightarrow \text{Verify Spent Coin} :$
then Adv fail

~~if~~
 $DS \leftarrow d_i \mid \exists i' > i : \left. \begin{array}{l} \text{Identify Double Spender} \\ (\text{spent coin } i, \\ \text{spent coin } i') \\ = \text{User } j \end{array} \right\}$

if $DS \neq \emptyset$ 「 #DS $\leftarrow d_i$ 」
then Adv fail

Adv succeeded.

In our system :

|| Possible since no secret
information of Alice is built
into the coins.

Possible
repair!
Add a signature
of Alice
on id + coins
to the used U.

Experiment ANONYMITY

e€
4.04.08

7

Adversary executes Bank Setup
and Bank Key Generation.

Adversary with access to
oracle for User Key Generation,
oracle for User - Withdrawal,
oracle for Payment.

ad returns i_0, j_0, i_1, j_1 , merchant id,
transaction id
b Ex 13. \leftarrow Payment(coin # i_b , merchant id, transaction id,
bank public key,
user secret key # j_b ,
coin secret(# j_s , # i_b))
spent coin

i.e. one of two predetermined
coins i_0, i_1 is spent
by user j_0 or user j_1

Adversary with access to
oracle for Payment
tries to figure out who spent that coin,
say it answers d.

If Adversary did query one of the
two payments in question

then Adv fail. \longrightarrow

If $d \neq b$ then Adv fail \longrightarrow

Adv succeeded.

Experiment EXCULPABILITY

cE
4.04.08
(8)

... The bank should not be able
to create proofs of withdrawal,
ie. coins which the user cannot spend.



Jens
(not shown)

Daniel

Matthias

Malte

Tim

Damian

Benedikt

Kathrin

Thomas

Henning

Roland

Shahram

Christoph

Andreas

Nico

Michael