Cryptographic e-Cash

Jan Camenisch

IBM Research – Zurich
@JanCamenisch
ibm.biz/jancamenisch
eCash scenario & requirements

Requirements
- Anonymity: Withdrawal and Deposit must be unlinkable
- No Double Spending: Coin is bit-strings, can be spend twice
Towards a Solution: do it like paper money

- Sign notes with digital signature scheme
  - Note = (serial number #, value)
  - Secure because
    * signature scheme can not be forged
    * bank will accepts some serial number only once → on-line e-cash
  - *Not* anonymous because (cf. paper solution)
    * bit-string of signature is unique
    * serial number is unique
Towards a Solution

- Use (more) cryptography
  - Hide serial number from bank when issuing
    - e.g., sign commitment of serial number
  - Reveal serial number and proof
    - knowledge of signature on
    - commitment to serial number
  - Anonymous because of commitments scheme and zero-knowledge proof
How to implement this?

---

… challenge is to do all this efficiently!
mathematical setting
A set G with operation $\diamond$ is called a group if:

- closure
  
  for all $a,b$, in $G \rightarrow a \diamond b$ in $G$

- commutativity
  
  for all $a,b$, in $G \rightarrow a \diamond b = b \diamond a$

- associativity
  
  for all $a,b,c$, in $G \rightarrow (a \diamond b) \diamond c = a \diamond (b \diamond c)$

- identity
  
  there exist some $e$ in $G$, s.t. for all $a$: $a \diamond e = a$

- invertibility
  
  for all $a$ in $G$, there exist $a^{-1}$ in $G$: $a \diamond a^{-1} = e$

- Example:
  
  integers under addition $(Z,+)=\{..., -2, -1, 0, 1, 2, ...\}$ or $(Zn,+)=\{0, 1, 2, ..., n-1\}$
  identity: $e = 0$
  inverse: $a^{-1} = -a$
Cyclic Groups

- **exponentiation** = repeated application of \( \cdot \), e.g., \( a^3 = a \cdot a \cdot a \)

- a group is **cyclic** if every element is power of some fixed element:
  - i.e., for each \( a \) in \( G \), there is unique \( i \) such that \( g^i = a \)
  - \( g \) = generator of the group
  - define \( g^0 = 1 \) = identity element
  - \( G = \langle g \rangle = \{1=g^0, g^1, g^2, \ldots, g^{q-1}\} \)
  - \( q = |G| \) = order of group
    - if \( q \) is a prime number then \( G \) is cyclic
    - computation in exponents can be done **modulo** \( q \):
      \[
g^i = g^{i \mod q}
\]

- computing with exponents:
  \[
g^{i+j} = g^i \cdot g^j \quad \quad g^{i\cdot j} = g^i / g^j = g^i \cdot (g^j)^{-1}
\]
  \[
g^{ij} = (g^i)^j \quad \quad g^{-i} = (g^{-1})^i = (g^i)^{-1}
\]
The Discrete Logarithm Problem

given $g$ and $x$ it is **easy** to compute $g^x, g^{1/x}, \ldots$

given $g^x$ and $g^y$ it is **easy** to compute $g^x g^y = g^{x+y}$

**Discrete Log Assumption**

given $g^x$ it is hard to *compute* $x$

**Diffie-Hellman Assumption**

given $g^x$ and $g^y$ it is hard to *compute* $g^{xy}$

**Decisional Diffie-Hellman Assumption**

given $g^x, g^y, \text{ and } g^z$ it is hard to *decide* if $g^z = g^{xy}$
commitment scheme
Commitment Scheme: Functionality

\[ m \rightarrow \text{Box} \rightarrow m \]

\[ m, 2-36-17 \rightarrow \text{Box} \rightarrow m? \]
Binding

$\text{m, 2-36-17}$

$\text{m', 3-21-11}$

$m \notin m'$

$m' \notin m$
Commitment Scheme: Security

Binding

m, 2-36-17

m', 3-21-11

m, m' 

m'?
Hiding: for all message $m, m'$
Hiding: for all message $m, m'$
Commitment Schemes

Group $G = \langle g \rangle = \langle h \rangle$ of order $q$

To commit to element $x \in \mathbb{Z}_q$:

- **Pedersen**: perfectly hiding, computationally binding
  
  choose $r \in \mathbb{Z}_q$ and compute $c = g^x h^r$

- **ElGamal**: computationally hiding, perfectly binding:
  
  choose $r \in \mathbb{Z}_q$ and compute $c = (g^x h^r, g^r)$

To open commitment:

- reveal $x$ and $r$ to verifier

- verifier checks if $c = g^x h^r$
Pedersen's Scheme:

Choose \( r \in \mathbb{Z}_q \) and compute \( c = g^x h^r \)

Perfectly hiding:

Let \( c \) be a commitment and \( u = \log_g h \)

Thus \( c = g^x h^r = g^{x+ur} = g^{(x+ur')+(r-r')} \)

\[ = g^{x+ur'} h^{r-r'} \quad \text{for any } r'! \]

I.e., given \( c \) and \( x' \) here exist \( r' \) such that \( c = g^{x'} h^{r'} \)

Computationally binding:

Let \( c, (x', r') \) and \( (x, r) \) s.t. \( c = g^{x'} h^{r'} = g^x h^r \)

Then \( g^{x'-x} = h^{r-r'} \) and \( u = \log_g h = (x'-x)/(r-r') \mod q \)
Proof of Knowledge of Contents

Proof of Relations among Contents
Commitment Scheme: Extended Features

Let $C_1 = g^m h^r$ and $C' = g^{m'} h^r$ then:

$\text{PK}\{ (\alpha, \beta) : \ C = g^\beta h^\alpha \}$

$\text{PK}\{ (\alpha, \beta, \gamma) : \ C' = g^\beta h^\alpha \land C = (g^2)^\beta h^\gamma \}$
Zero-Knowledge Proofs

- interactive proof between a prover and a verifier about the prover's knowledge

- properties:
  - **zero-knowledge**
    verifier learns nothing about the prover's secret
  - **proof of knowledge (soundness)**
    prover can convince verifier only if she knows the secret
  - **completeness**
    if prover knows the secret she can always convince the verifier
Given group \( <g> \) and element \( y \in <g> \).

Prover wants to convince verifier that she *knows* \( x \) s.t. \( y = g^x \) such that verifier only learns \( y \) and \( g \).

**Prover:**

- Choose a random \( r \)
- Compute \( t := g^r \)
- Compute \( s := r - cx \)

**Verifier:**

- Compute \( t \)
- Compute \( c \)
- Compute \( t = g^s y^c \) ?

**Notation:** \( PK\{ (\alpha): \ y = g^\alpha \} \)
**Proof of Knowledge Property:**

If prover is successful with non-negl. probability, then she “knows” \( x = \log g^y \), i.e., ones can extract \( x \) from her.

Assume \( c \in \{0,1\}^k \) and consider execution tree:

If success probability for any prover (including malicious ones) is \( >2^{-k} \) then there are two *accepting* tuples \((t,c_1,s_1)\) and \((t,c_2,s_2)\) for the same \( t \).
Zero Knowledge Proofs: Security

Prover might do protocol computation in any way it wants & we cannot analyse code.

Thought experiment:
- Assume we have prover as a black box → we can reset and rerun prover
- Need to show how secret can be extracted via protocol interface

\[
\begin{align*}
t &= g^s y^c = g^{s'} y^{c'} \\
\rightarrow & \quad y^{c' - c} = g^{s - s'} \\
\rightarrow & \quad y = g^{(s - s')/(c' - c)} \\
\rightarrow & \quad x = (s - s')/(c' - c) \mod q
\end{align*}
\]
Zero-Knowledge Proofs: Security

Zero-knowledge property:
If verifier does not learn anything (except the fact that Alice knows $x = \log g y$)

Idea: One can simulate whatever Bob “sees”.

Choose random $c', s'$
compute $t := g^{s'} y^{c'}$

if $c = c'$ send $s' = s$, otherwise restart

Problem: if domain of $c$ too large, success probability becomes too small
One way to modify protocol to get large domain $c$:

Prover:
- random $r$
- $t := g^r$
- $h := H(c,v)$
- $s := r - cx$

Verifier:
- random $c,v$
- $h := H(c,v)$
- $t = g^s y^c$?

notation: $PK\{ (\alpha) : y = g^\alpha \}$
One way to modify protocol to get large domain $c$:

Choose random $c', s'$
compute $t' := g^{s'} y^{c'}$

after having received $c$ “reboot” verifier

Choose random $s$
compute $t := g^s y^c$
send $s$
Signature \( SPK\{(a): \ y = g^a \}(m): \)

Signing a message \( m \):  
- chose random \( r \in Z_q \) and  
- compute \( c := H(g^r \| m) = H(t \| m) \)  
  \( s := r - cx \mod (q) \)  
- output \( (c,s) \)

Verifying a signature \( (c,s) \) on a message \( m \):  
- check \( c = H(g^s y^c \| m) ? \leftrightarrow t = g^s y^c ? \)

Security:  
- underlying protocol is zero-knowledge proof of knowledge  
- hash function \( H(.) \) behaves as a “random oracle.”
Zero Knowledge Proofs of Knowledge of Discrete Logarithms

Many Exponents:

\[ \text{PK}((\alpha,\beta,\gamma,\delta): \ y = g^\alpha h^\beta z^\gamma k^\delta u^\beta } \]

Logical combinations:

\[ \text{PK}((\alpha,\beta): \ y = g^\alpha \land z = g^\beta \land u = g^\beta h^\alpha } \]
\[ \text{PK}((\alpha,\beta): \ y = g^\alpha \lor z = g^\beta } \]

Intervals and groups of different order (under SRSA):

\[ \text{PK}((\alpha): \ y = g^\alpha \land \alpha \in [A,B] } \]
\[ \text{PK}((\alpha): \ y = g^\alpha \land z = g^\alpha \land \alpha \in [0,\min\{\text{ord}(g),\text{ord}(g)\}] } \]

Non-interactive (Fiat-Shamir heuristic, Schnorr Signatures):

\[ \text{PK}((\alpha): \ y = g^\alpha } (m) \]
Some Example Proofs and Their Analysis

Let $g, h, C_1, C_2, C_3$ be group elements.

Now, what does

$$\text{PK}\{ (\alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3, \beta_3) : \quad C_1 = g^{\alpha_1} h^{\beta_1} \land C_2 = g^{\alpha_2} h^{\beta_2} \land C_3 = g^{\alpha_3} h^{\beta_3} \land C_3 = g^{\alpha_1 + \alpha_2} h^{\beta_3} \}$$

mean?

→ Prover knows values $\alpha_1, \beta_1, \alpha_2, \beta_2, \beta_3$ such that

$$C_1 = g^{\alpha_1} h^{\beta_1}, \quad C_2 = g^{\alpha_2} h^{\beta_2} \text{ and } \quad C_3 = g^{\alpha_1 + \alpha_2} h^{\beta_3} = g^{\alpha_3} h^{\beta_3}$$

$\alpha_3 = \alpha_1 + \alpha_2 \pmod{q}$

And what about:

$$\text{PK}\{ (\alpha_1, \ldots, \beta_3) : \quad C_1 = g^{\alpha_1} h^{\beta_1} \land C_2 = g^{\alpha_2} h^{\beta_2} \land C_3 = g^{\alpha_3} h^{\beta_3} \land C_3 = g^{\alpha_1 (g^5) a_2} h^{\beta_3} \}$$

→ $C_3 = g^{\alpha_1 + 5 \alpha_2} h^{\beta_3}$

$\alpha_3 = \alpha_1 + 5 \alpha_2 \pmod{q}$
Let \( g, h, C_1, C_2, C_3 \) be group elements.

Now, what does
\[
PK\{(\alpha_1, \ldots, \beta_3):\quad C_1 = g^{\alpha_1} h^{\beta_1} \land C_2 = g^{\alpha_2} h^{\beta_2} \land C_3 = g^{\alpha_3} h^{\beta_3} \land C_3 = C_2^{\alpha_1} h^{\beta_3}\}
\]
mean?

→ Prover knows values \( \alpha_1, \beta_1, \alpha_2, \beta_2, \beta_3 \) such that
\[
C_1 = g^{\alpha_1} h^{\beta_1}, \quad C_2 = g^{\alpha_2} h^{\beta_2} \quad \text{and}
\]
\[
C_3 = C_2^{\alpha_1} h^{\beta_3} = (g^{\alpha_2} h^{\beta_2})^{\alpha_1} h^{\beta_3} = g^{\alpha_2 \cdot \alpha_1} h^{\beta_3 + \beta_2 \cdot \alpha_1}
\]
\[
C_3 = g^{\alpha_2 \cdot \alpha_1} h^{\beta_3 + \beta_2 \cdot \alpha_1} = g^{\alpha_3} h^{\beta_3'}
\]
\[
a_3 = a_1 \cdot a_2 \pmod{q}
\]

And what about
\[
PK\{(\alpha_1, \beta_1, \beta_2):\quad C_1 = g^{\alpha_1} h^{\beta_1} \land C_2 = g^{\alpha_2} h^{\beta_2} \land C_2 = C_1^{\alpha_1} h^{\beta_2}\}
\]

→ \( a_2 = a_1^2 \pmod{q} \)
Some Example Proofs and Their Analysis

Let $g, h, C_1, C_2, C_3$ be group elements.

Now, what does
\[ \text{PK}((\alpha_1, \ldots, \beta_2): \quad C_1 = g^{\alpha_1} h^{\beta_1} \land C_2 = g^{\alpha_2} h^{\beta_2} \land g = (C_2/C_1)^{\alpha_1} h^{\beta_2} } \]
mean?

→ Prover knows values $\alpha, \beta_1, \beta_2$ such that

\[ C_1 = g^{\alpha_1} h^{\beta_1} \]
\[ g = (C_2/C_1)^{\alpha_1} h^{\beta_2} = (C_2 g^{-\alpha_1} h^{-\beta_1})^{\alpha_1} h^{\beta_2} \]

→
\[ g^{1/\alpha_1} = C_2 g^{-\alpha_1} h^{-\beta_1} h^{\beta_2/\alpha_1} \]
\[ C_2 = g^{\alpha_1} h^{\beta_2} a_1 + g^{1/\alpha_1} h^{\beta_1 - \beta_2/\alpha_1} \]
\[ C_2 = g^{\alpha_2} h^{\beta_2} \]
\[ a_2 = a_1 + a_1^{-1} \pmod{q} \]
signature schemes
Key Generation
Signature Scheme: Functionality

\[ \sigma = \text{sig}((m_1, \ldots, m_k)) \]
Signature Scheme: Functionality

\[ \sigma = \text{sig}((m_1, ..., m_k)) \]

\[ \text{ver}(\sigma, (m_1, ..., m_k)) = \text{true} \]
Signature Scheme: Security

Unforgeability under Adaptive Chosen Message Attack

$m_1 \sigma_1$
Unforgeability under Adaptive Chosen Message Attack
Signature Scheme: Security

Unforgeability under Adaptive Chosen Message Attack

\[
\sigma' \text{ and } m' \neq m_i \text{ s.t. } 
\text{ver}(\sigma', m', \sigma') = \text{true}
\]
Unforgeability under Adaptive Chosen Message Attack

Signature Scheme: Security

\[ \sigma' \text{ and } m' \neq m_i \Rightarrow \text{ver}(\sigma', m', \sigma_i) = \text{true} \]
signature schemes with protocols
Signature Scheme: Signing Hidden Messages

\[ \sigma = \text{sig}(\langle m_1, \ldots, m_j, m_{j+1}, \ldots, m_k \rangle, \varphi) \]

\[ \text{ver}(\sigma, (m_1, \ldots, m_k), \varphi) = \text{true} \]

Verification remains unchanged!

Security requirements basically the same, but

- Signer should not learn any information about \( m_1, \ldots, m_j \)
- Forgery w.r.t. message clear parts and opening of commitments
Proving Possession of a Signature

\[ \sigma \text{ on } (m_1, ..., m_k) \]
Proving Possession of a Signature

\[ \sigma \text{ on } (m_1, \ldots, m_k) \]

\[ \{m_i \mid i \in S\} \]
Proving Possession of a Signature

Variation:
- Send also \( m_i \) to verifier and
- Prove that committed messages are signed
- Prove properties about hidden/committed \( m_i \)
Blind Signatures vs Signature with Protocols

- Can be used multiple times
  - Damgaard, Camenisch & Lysyanska ya
  - Strong RSA, DL-ECC, ...

- Can be used only once
  - Chaum, Brands, et al.
  - Discrete Logs, RSA, ..
Some signature schemes
RSA Signature Scheme – For Reference

Rivest, Shamir, and Adlemann 1978

Secret Key: two random primes $p$ and $q$
Public Key: $n := pq$, prime $e$, and collision-free hash function

$H: \{0,1\}^* \rightarrow \{0,1\}^\ell$

Computing signature on a message $m \in \{0,1\}^*$

$$d := 1/e \mod (p-1)(q-1)$$
$$s := H(m)^d \mod n$$

Verification of signature $s$ on a message $m \in \{0,1\}^*$

$$s^e = H(m) \pmod{n}$$

Correctness: $s^e = (H(m)^d)^e = H(m)^{d\cdot e} = H(m) \pmod{n}$
Verification signature on a message $m \in \{0,1\}^*$

$$s^e := H(m) \pmod{n}$$

Wanna do proof of knowledge of signature on a message, e.g.,

$$PK\{ (m,s): s^e = H(m) \pmod{n} \}$$

But this is not a valid proof expression!!!! :-(

Public key of signer: RSA modulus $n$ and $a_i, b, d \in QR_n$.

Secret key: factors of $n$

To sign $k$ messages $m_1, \ldots, m_k \in \{0,1\}^\ell$:

- choose random prime $2^{\ell+2} > e > 2^{\ell+1}$ and integer $s \approx n$
- compute $c$:
  $$c = \left(\frac{d}{(a_1^{m_1} \cdots a_k^{m_k} b^s)}\right)^{1/e} \mod n$$
- signature is $(c,e,s)$
To verify a signature \((c,e,s)\) on messages \(m_1, \ldots, m_k\):

- \(m_1, \ldots, m_k \in \{0,1\}^\ell\):
- \(e > 2^{\ell+1}\)
- \(d = c^e a_1^{m_1} \cdots a_k^{m_k} b^s \mod n\)

Theorem: *Signature scheme is secure against adaptively chosen message attacks under Strong RSA assumption.*
Sign blindly with CL signatures

\[ \sigma = \text{sig}((m_1, \ldots, m_j, m_{j+1}, \ldots, m_k), \hat{c}) \]

Choose \( e, s'' \)

\[ c = (d/(C a_3^{m_3} b^{s''}))^{1/e} \mod n \]

\[ d = c^e a_1^{m_1} a_2^{m_2} a_3^{m_3} b^{s''+s''} \mod n \]

\[ C = a_1^{m_1} a_2^{m_2} b^{s'} \]

\[ C + \text{PK}((m_1, m_2, s')): C = a_1^{m_1} a_2^{m_2} b^{s'} \]
Proving Knowledge of a CL-signature

Recall: \[ d = c^e a_1^{m_1} a_2^{m_2} b^s \mod n \]

Observe:

- Let \( c' = c b^{\dagger} \mod n \) with randomly chosen \( \dagger \)
- Then \( d = c'^e a_1^{m_1} a_2^{m_2} b^{s-e^\dagger} \mod n \), i.e., \((c', e, s^* = s-e^\dagger)\) is also signature on \( m_1 \) and \( m_2 \)

To prove knowledge of signature \((c', e, s^*)\) on \( m_2 \) and some \( m_1 \)
- provide \( c' \)
- \( PK\{ (\epsilon, \mu_1, \sigma) : \; d/a_2^{m_2} := c'^\epsilon a_1^{\mu_1} b^\sigma \land \mu \in \{0,1\}^\ell \land \epsilon > 2^{\ell+1} \} \)

\[ \rightarrow \text{proves } d := c'^\epsilon a_1^{\mu_1} a_2^{m_2} b^\sigma \]
Realizing On-Line eCash
Recall basic idea

- Issue coin: Hide serial number from bank when issuing
  - sign commitment of random serial number

- Spend coin: reveal serial number and proof
  - knowledge of signature on
  - commitment to serial number
On-line E-cash: Withdrawal

Choose \( e, s'' \)

\[ c = \left( \frac{d}{(C b^{s''})^{1/e}} \right) \mod n \]

Choose random \( #, s' \)

and compute

\[ C = a_1^# b^{s'} \]

\((c,e,s'') + s'\) s.t.

\[ d = c^e a_1^# b^{s''} + s' \mod n \]
(c,e,s''+s') s.t.
\[ d = c^e a_1^{\#} b^{s''} + s' \pmod{n} \]

compute
\[ c' = c b^{s'} \pmod{n} \]

proof = \( PK\{ (\epsilon, \mu, \rho, \sigma) : \quad d / a_1^{\#} = c' \epsilon b^\sigma \pmod{n} \} \)
On-line E-cash: Payment

\[(c,e,s''+s') \text{ s.t. } d = c^e a_1 \# b^{s''} + s' \pmod{n}\]

compute
\[c' = c b^{s'} \pmod{n}\]

proof = \(PK\{(\epsilon, \mu, \rho, \sigma) : d / a_1 \# = c' \epsilon b^\rho (\pmod{n}) \}\)
Anonymity
- Bank does not learn # during withdrawal
- Bank & Shop do not learn c, e when spending
Double Spending:

- **Spending = Compute**
  \[-c' = c \cdot b^{s'} \mod n\]
  
  - **proof = PK\{(\varepsilon, \mu, \rho, \sigma) : \ d / a_1^\# = c' \cdot \varepsilon \cdot b^\sigma (\mod n) \}**

- **Can use the same # only once....**
  
  - If more #'s are presented than withdrawals:
    
    - Proofs would not sound
    
    - Signature scheme would not secure
Realizing Off-Line eCash
Recall On-Line E-Cash

On-Line Solution:
1. Coin = random serial # (chosen by user) signed by Bank
2. Banks signs blindly
3. Spending by sending # and prove knowledge of signature to Merchant
4. Merchant checks validy w/ Bank
5. Bank accepts each serial # only once.

Off-Line:
- Can check serial # only after the fact
- … but at that point user will have been disappeared...
Goal:

- spending coin once: OK
- spending coin twice: anonymity revoked

Seems like a paradox, but crypto is all about solving paradoxical problems :-)
Main Idea:

- Include #, id, r
- Upon spending:
  reveal #, and \( t = id + r \cdot u \),
  with \( c \) randomly chosen by merchant
- \( t \) won't reveal anything about id!
- However, given two equations (for the same #, id, r)
  \( t1 = id + r \cdot u1 \)
  \( t2 = id + r \cdot u2 \)
  one can solve for id.
Off-line E-cash: Withdrawal

choose random $\#, r, s'$
and compute

$C = a_1^\# a_2^r b^{s'}$

$(c,e,s'' + s')$ s.t.

$d = c^e a_1^\# a_2^r a_3^\text{nym} b^{s''} + s' \pmod{n}$
Let $G=<g>$ be a group of order $q$

$$(c, e, s''+s') \text{ s.t.}$$

$$d = c^e a_1 # a_2 r \text{nym} a_3 b^{s''} + s' \pmod{n}$$

compute

$$t = r + u \text{nym} \pmod{q}$$

$$c' = c \text{ b}^{s'} \pmod{n}$$

proof = $PK\{(\varepsilon, \mu, \rho, \sigma) :$

$$d / a_1 ^# = c' \varepsilon a_2 ^\rho a_3 ^\mu b ^\sigma \pmod{n} \land g'^t = g^\rho (g^u)^\mu \}$$
Off-line E-cash: Payment

PK\{(ε, μ, ρ, σ) : \\
\frac{d}{a_1} \# = c'ε a_2^\rho a_3^\mu b^\sigma \pmod{n} \land g^\dagger = g^\rho (g^\mu)^\mu\}

1. \(d = c'ε a_1^\# a_2^\rho a_3^\mu b^\sigma \pmod{n}\) \\
   \implies (c', ε, σ) is a signature on (\#, μ, ρ)

2. \(g^\dagger = g^{\rho + \mu μ}\) \\
   \implies \dagger = ρ + u μ \pmod{q}, \\
i.e., \dagger was computed correctly!
Off-line E-cash: Deposit

# $\in L$?

If so:
1. $t = \rho + u \mu \quad (\text{mod } q)$
2. $t' = \rho + u' \mu \quad (\text{mod } q)$

solve for $\rho$ and $\mu$.

$\Rightarrow \mu = nym$ because of proof

u, t, #, proof
Off-line E-cash: Security

- Unforgeable:
  - no more coins than #',
    - otherwise one can forge signatures
    - or proofs are not sound
  - if coins with same # appears with different u's => reveals nym

- Anonymity:
  - # and r are hidden from signer upon withdrawal
  - t does not reveal anything about nym (is blinded by r)
  - proof proof does not reveal anything
Extensions and more

e-Cash

- K-spendable cash
  - Multiple serial numbers and randomizers per coin
  - Use PRF to generate serial number and randomizers from seed in coin

- Money laundering preventions
  - Must not spend more than $10000 dollars with same party
  - Essentially use additional coin defined per merchant that controls this

Other protocols from these building blocks, essentially anything with authentication and privacy

- Anonymous credentials, eVoting, ....

Alternative building blocks

- A number of signatures scheme that fit the same bill
- (Verifiable) encryption schemes that work along as well
- Alternative framework: Groth-Sahai proofs plus “structure-preserving” schemes
Thank you!

- eMail: identity@zurich.ibm.com
- Links:
  - www.abc4trust.eu
  - www.futureID.eu
  - www.au2eu.eu
  - www.PrimeLife.eu
  - www.zurich.ibm.com/idemix
  - idemixdemo.zurich.ibm.com
- Code
  - github.com/p2abcengine & abc4trust.eu/idemix


Mihir Bellare: Computational Number Theory
http://www-cse.ucsd.edu/~mihir/cse207/w-cnt.pdf


Jan Camenisch, Maria Dubovitskaya, Gregory Neven: Oblivious transfer with access control. ACM Conference on Computer and Communications Security 2009: 131-140


References


- Camenisch, Shoup: Practical Verifiable Encryption and Decryption of Discrete Logarithms. CRYPTO 2003: 126-144


- D. Chaum: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. In Communications of the ACM.


- T. ElGamal: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In Advances in Cryptology - CRYPTO '84.