Non-boring cryptography

Christian Grothoff

Berner Fachhochschule

March 31, 2018
Learning Objectives

- Blind signatures and applications
- Homomorphic encryption
- Secure multiparty computation (SMC) theory
- Common SMC adversary models
- Specialized protocols for SMC problems
Reminder: RSA

Pick $p, q$ prime and $e$ such that

$$GCD((p - 1)(q - 1), e) = 1 \quad (1)$$

- Define $n = pq$,
- compute $d$ such that $ed \equiv 1 \mod (p - 1)(q - 1)$.
- Let $s := m^d \mod n$.
- Then $m \equiv s^e \mod n$. 
RSA Summary

- Public key: \( n, e \)
- Private key: \( d \equiv e^{-1} \mod \phi(n) \) where \( \phi(n) = (p - 1) \cdot (q - 1) \)
- Encryption: \( c \equiv m^e \mod n \)
- Decryption: \( m \equiv c^d \mod n \)
- Signing: \( s \equiv m^d \mod n \)
- Verifying: \( m \equiv s^e \mod n ? \)
Low Encryption Exponent Attack

- $e$ is known
- $M$ maybe small
- $C = M^e < n$?
- If so, can compute $M = \sqrt[e]{C}$

$\Rightarrow$ Small $e$ can be bad!
Padding and RSA Symmetry

- Padding can be used to avoid low exponent issues (and issues with $m = 0$ or $m = 1$)
- Randomized padding defeats chosen plaintext attacks
- Padding breaks RSA symmetry:
  \[ D_{A_{priv}}(D_{B_{priv}}(E_{A_{pub}}(E_{B_{pub}}(M)))) \neq M \]  (2)

- PKCS#1 / RFC 3447 define a padding standard
1. Obtain public key
   \((e, n)\)
2. Compute
   \(f := \text{FDH}(m),\)
   \(f < n.\)
3. Pick blinding factor
   \(b \in \mathbb{Z}_n\)
4. Transmit
   \(f' := fb^e \mod n\)
Blind signatures with RSA

1. Obtain public key 
   \((e, n)\)

2. Compute 
   \(f := FDH(m), \quad f < n.\)

3. Pick blinding factor 
   \(b \in \mathbb{Z}_n\)

4. Transmit 
   \(f' := fb^e \mod n\)

1. Receive \(f'\).

2. Compute 
   \(s' := f'^d \mod n.\)

3. Send \(s'\).
Blind signatures with RSA

1. Obtain public key \((e, n)\)
2. Compute \(f := \text{FDH}(m), f < n\).
3. Pick blinding factor \(b \in \mathbb{Z}_n\)
4. Transmit \(f' := fb^e \mod n\)

1. Receive \(f'\).
2. Compute \(s' := f'^d \mod n\).
3. Send \(s'\).

1. Receive \(s'\).
2. Compute \(s := s'b^{-1} \mod n\)
Break
A Social Problem

This was a question posed to RAND researchers in 1971:

“Suppose you were an advisor to the head of the KGB, the Soviet Secret Police. Suppose you are given the assignment of designing a system for the surveillance of all citizens and visitors within the boundaries of the USSR. The system is not to be too obtrusive or obvious. What would be your decision?”
A Social Problem

This was a question posed to RAND researchers in 1971:

“Suppose you were an advisor to the head of the KGB, the Soviet Secret Police. Suppose you are given the assignment of designing a system for the surveillance of all citizens and visitors within the boundaries of the USSR. The system is not to be too obtrusive or obvious. What would be your decision?”

Mastercard/Visa are too transparent.

“I think one of the big things that we need to do, is we need to get a way from true-name payments on the Internet. The credit card payment system is one of the worst things that happened for the user, in terms of being able to divorce their access from their identity.”

The Bank’s Problem

3D secure ("verified by visa") is a nightmare:

- Complicated process
- Shifts liability to consumer
- Significant latency
- Can refuse valid requests
- Legal vendors excluded
- No privacy for buyers

Online credit card payments will be replaced, but with what?
The Bank’s Problem

- Global tech companies push oligopolies
- Privacy and federated finance are at risk
- Economic sovereignty is in danger
Do you want to live under total surveillance?
Digital cash, made socially responsible.

Privacy-Preserving, Practical, Taxable, Free Software, Efficient
What is Taler?

Taler is an electronic instant payment system.

- Uses electronic coins stored in wallets on customer’s device
- Like cash
- Pay in existing currencies (i.e. EUR, USD, BTC), or use it to create new regional currencies
Taler Overview

Exchange

withdraw coins

Customer

spend coins

Merchant

deposit coins
Architecture of Taler

1. pay exchange
2. wire transfer
3. withdraw coins
4. spend coins
5. deposit coins
6. wire transfer
7. view balance

⇒ Convenient, taxable, privacy-enhancing, & resource friendly!
Usability of Taler

[9x251]Usability of Taler

https://demo.taler.net/

1. Install Chrome extension.
2. Visit the bank.demo.taler.net to withdraw coins.
3. Visit the shop.demo.taler.net to spend coins.
We say Taler is taxable because:

- Merchant’s income is visible from deposits.
- Hash of contract is part of deposit data.
- State can trace income and enforce taxation.
We say Taler is taxable because:

- Merchant’s income is visible from deposits.
- Hash of contract is part of deposit data.
- State can trace income and enforce taxation.

Limitations:

- withdraw loophole
- *sharing* coins among family and friends
How does it work?

We use a few ancient constructions:

- Cryptographic hash function (1989)
- Blind signature (1983)
- Schnorr signature (1989)
- Diffie-Hellman key exchange (1976)
- Cut-and-choose zero-knowledge proof (1985)

But of course we use modern instantiations.
Exchange setup: Create a denomination key (RSA)

1. Pick random primes $p, q$.
2. Compute $n := pq$, 
   $\phi(n) = (p - 1)(q - 1)$
3. Pick small $e < \phi(n)$ such that $d := e^{-1} \mod \phi(n)$ exists.
4. Publish public key $(e, n)$. 
Merchant: Create a signing key (EdDSA)

- pick random $m \mod o$ as private key
- $M = mG$ public key

Capability: $m \Rightarrow M$
Customer: Create a planchet (EdDSA)

- Pick random $c \mod o$ private key
- $C = cG$ public key

Capability: $c \Rightarrow$
1. Obtain public key $(e, n)$
2. Compute $f := FDH(C)$, $f < n$.
3. Pick blinding factor $b \in \mathbb{Z}_n$
4. Transmit $f' := fb^e \mod n$
Exchange: Blind sign (RSA)

1. Receive $f'$.
2. Compute $s' := f'^d \mod n$.
3. Send signature $s'$.
Customer: Unblind coin (RSA)

1. Receive $s'$.
2. Compute $s := s' b^{-1} \mod n$
Withdrawing coins on the Web

Taler (Withdraw coins)

Customer Browser

Bank Site

Taler Exchange

HTTPS

HTTPS

wire transfer

1 user authentication
2 send account portal
3 initiate withdrawal (specify amount and exchange)
4 request coin denomination keys and wire transfer data
5 send coin denomination keys and wire transfer data
6 execute withdrawal

opt
7 request transaction authorization
8 transaction authorization

9 withdrawal confirmation

10 execute wire transfer

11 withdraw request
12 signed blinded coins
13 unblind coins

Customer Browser

Bank Site

Taler Exchange
Customer: Build shopping cart
Merchant Integration: Wallet Detection

```html
<script src="taler-wallet-lib.js"></script>
<script>
  taler.onPresent(() => {
    alert("Taler wallet is installed");
  });
  taler.onAbsent(() => {
    alert("Taler wallet is not installed");
  });
</script>
```
HTTP/1.1 402 Payment Required
Content-Type: text/html; charset=UTF-8
X-Taler-Contract-Url: https://shop/generate-contract/42

<!DOCTYPE html>
<html>
   <!-- fallback for browsers without the Taler extension -->
   You do not seem to have Taler installed, here are other payment options ...
</html>
Merchant Integration: Contract

{ "H_wire": "YTH0C4QBCQ10VDNTJN0DCTTTV2Z6JHT5NF43F0QRHZ8JYB5NG4W4...", "amount": { "currency": "EUR", "fraction": 0, "value": 1 }, "max_fee": { "currency": "EUR", "fraction": 100000, "value": 0 }, "auditors": [{ "auditor_pub": "42V6TH91Q83FB846DK1GW3JQ5E8DS273W4..." }], "exchanges": [{ "master_pub": "1T5FA8VQHMMKBHDMYPRZA2ZF2S63AKF0Y...", "url": "https://exchange/" }], "fulfillment_url": "https://shop/article/42?tid=249&time=14714744", "merchant": { "address": "Mailbox 4242", "jurisdiction": "Jersey", "name": "Shop Inc." }, "merchant_pub": "Y1ZAR5346J3ZTEXJCHQY9NJN78EZ2HSKZK8M0MYTNJRG5N...", "products": [{ "description": "Essay: The GNU Project", "price": { "currency": "EUR", "fraction": 0, "value": 1 }, "product_id": 42, "quantity": 1 }, "pay_deadline": "/Date(1480119270)/", "refund_deadline": "/Date(1471522470)/", "timestamp": "/Date(1471479270)/", "transaction_id": 249960194066269 }
Merchant: Propose contract (EdDSA)

1. Complete proposal $D$.
2. Send $D, EdDSA_m(D)$
Customer: Spend coin (EdDSA)

1. Receive proposal $D$, $EdDSA_m(D)$.
2. Send $s$, $C$, $EdDSA_c(D)$
Merchant and Exchange: Verify coin (RSA)

\[ s^e \equiv m \mod n \]
Payment processing with Taler

1. Choose goods by navigating to offer URL
2. Send signed digital contract proposal
3. Select Taler payment method (skippable with auto-detection)
4. Affirm contract
5. Navigate to fulfillment URL
6. Send hash of digital contract and payment information
7. Send payment
8. Forward payment
9. Confirm payment
10. Confirm payment
11. Reload fulfillment URL for delivery
12. Provide product resource
Break
Giving change

It would be inefficient to pay EUR 100 with 1 cent coins!

- Denomination key represents value of a coin.
- Exchange may offer various denominations for coins.
- Wallet may not have exact change!
- Usability requires ability to pay given sufficient total funds.
Giving change

It would be inefficient to pay EUR 100 with 1 cent coins!
▶ Denomination key represents value of a coin.
▶ Exchange may offer various denominations for coins.
▶ Wallet may not have exact change!
▶ Usability requires ability to pay given sufficient total funds.

Key goals:
▶ maintain unlinkability
▶ maintain taxability of transactions
Giving change

It would be inefficient to pay EUR 100 with 1 cent coins!

- Denomination key represents value of a coin.
- Exchange may offer various denominations for coins.
- Wallet may not have exact change!
- Usability requires ability to pay given sufficient total funds.

Key goals:

- maintain unlinkability
- maintain taxability of transactions

Method:

- Contract can specify to only pay *partial value* of a coin.
- Exchange allows wallet to obtain *unlinkable change* for remaining coin value.
Diffie-Hellman (ECDH)

1. Create private keys $c$, $t$ mod $o$
2. Define $C = cG$
3. Define $T = tG$
4. Compute DH
   \[ cT = c(tG) = t(cG) = tC \]
Strawman solution

Given partially spent private coin key $c_{old}$:

1. Pick random $c_{new} \mod o$ private key
2. $C_{new} = c_{new} G$ public key
3. Pick random $b_{new}$
4. Compute $f_{new} := FDH(C_{new}), m < n.$
5. Transmit $f'_{new} := f_{new} b_{new}^e \mod n$

... and sign request for change with $c_{old}$.
Strawman solution

Given partially spent private coin key $c_{old}$:
1. Pick random $c_{new} \mod o$ private key
2. $C_{new} = c_{new} G$ public key
3. Pick random $b_{new}$
4. Compute $f_{new} := FDH(C_{new}), m < n.$
5. Transmit $f'_{new} := f_{new} b_{new}^e \mod n$
... and sign request for change with $c_{old}$.

Problem: Owner of $c_{new}$ may differ from owner of $c_{old}$!
Customer: Transfer key setup (ECDH)

Given partially spent private coin key $c_{old}$:

1. Let $C_{old} := c_{old}G$ (as before)
2. Create random private transfer key $t \mod o$
3. Compute $T := tG$
4. Compute $X := c_{old}(tG) = t(c_{old}G) = tC_{old}$
5. Derive $c_{new}$ and $b_{new}$ from $X$
6. Compute $C_{new} := c_{new}G$
7. Compute $f_{new} := FDH(C_{new})$
8. Transmit $f'_{new} := f_{new}b_{new}^{e}$
Cut-and-Choose

c_{old} \rightarrow t_{1} \rightarrow c_{new,1} \rightarrow b_{new,1} \rightarrow \text{transmit} \rightarrow \text{Exchange}

c_{old} \rightarrow t_{2} \rightarrow c_{new,2} \rightarrow b_{new,2} \rightarrow \text{transmit} \rightarrow \text{Exchange}

c_{old} \rightarrow t_{3} \rightarrow c_{new,3} \rightarrow b_{new,3} \rightarrow \text{transmit} \rightarrow \text{Exchange}
Exchange: Choose!

Exchange sends back random $\gamma \in \{1, 2, 3\}$ to the customer.
1. If $\gamma = 1$, send $t_2, t_3$ to exchange
2. If $\gamma = 2$, send $t_1, t_3$ to exchange
3. If $\gamma = 3$, send $t_1, t_2$ to exchange
Exchange: Verify ($\gamma = 2$)
Exchange: Blind sign change (RSA)

1. Take $f'_{\text{new}, \gamma}$.
2. Compute $s' := f'_{\text{new}, \gamma} \mod n$.
3. Send signature $s'$.
Customer: Unblind change (RSA)

1. Receive $s'$.
2. Compute $s := s' b_{new,\gamma}^{-1}$ mod $n$. 
Exchange: Allow linking change

Given $C_{old}$

return $T_\gamma$ and

$s := s' b_{new,\gamma}^{-1} \mod n$. 
Customer: Link (threat!)

1. Have $c_{old}$.
2. Obtain $T_\gamma, s$ from exchange
3. Compute $X_\gamma = c_{old} T_\gamma$
4. Derive $c_{new,\gamma}$ and $b_{new,\gamma}$ from $X_\gamma$
5. Unblind $s := s' b_{new,\gamma}^{-1}$ mod $n$
Refresh protocol summary

- Customer asks exchange to convert old coin to new coin
- Protocol ensures new coins can be recovered from old coin
  ⇒ New coins are owned by the same entity!

Thus, the refresh protocol allows:
- To give unlinkable change.
- To give refunds to an anonymous customer.
- To expire old keys and migrate coins to new ones.
- To handle protocol aborts.

Transactions via refresh are equivalent to sharing a wallet.
Secure Multiparty Computation (SMC)

Alice und Bob haben private Daten $a_i$ and $b_i$.

Alice und Bob führen ein Protokoll aus und berechnen gemeinsam $f(a_i, b_i)$.

Nur einer von beiden lernt das Ergebnis (i.d.R.)
Adversary model

Honest but curious
Homomorphic Encryption

\[ E(x_1 \oplus x_2) = E(x_1) \otimes E(x_2) \]
Multiplicative Homomorphism: RSA & ElGamal

▶ Unpadded RSA (multiplicative):

\[ E(x_1) \cdot E(x_2) = x_1^e x_2^2 = E(x_1 \cdot x_2) \]  

(4)

▶ ElGamal:

\[ E(x_1) \cdot E(x_2) = (g^{r_1}, x_1 \cdot h^{r_1})(g^{r_2}, x_2 \cdot h^{r_2}) \]  

\[ = (g^{r_1+r_2}, (x_1 \cdot x_2)h^{r_1+r_2}) \]  

\[ = E(m_1 \cdot m_2) \]  

(5)

(6)

(7)
Additive Homomorphism: Paillier

\[ E_K(m) := g^m \cdot r^n \mod n^2, \tag{8} \]

\[ D_K(c) := \frac{(c^\lambda \mod n^2) - 1}{n} \cdot \mu \mod n \tag{9} \]

where the public key \( K = (n, g) \), \( m \) is the plaintext, \( c \) the ciphertext, \( n \) the product of \( p, q \in \mathbb{P} \) of equal length, and \( g \in \mathbb{Z}_{n^2}^* \). In Paillier, the private key is \((\lambda, \mu)\), which is computed from \( p \) and \( q \) as follows:

\[ \lambda := \text{lcm}(p - 1, q - 1), \tag{10} \]

\[ \mu := \left( \frac{(g^\lambda \mod n^2) - 1}{n} \right)^{-1} \mod n. \tag{11} \]

Paillier offers additive homomorphic public-key encryption, that is:

\[ E_K(a) \otimes E_K(b) \equiv E_K(a + b) \tag{12} \]

for any public key \( K \).
Fully homomorphic encryption

Additive:

\[ E(A) \oplus E(B) = E(A + B) \]  \hspace{1cm} (13)

and multiplicative:

\[ E(A) \otimes E(B) = E(A \cdot B) \]  \hspace{1cm} (14)

Known cryptosystems: Brakerski-Gentry-Vaikuntanathan (BGV), NTRU, Gentry-Sahai-Waters (GSW).
Break
Example: Secure Scalar Product

- Original idea by Ioannids et al. in 2002 (use: \((a - b)^2 = a^2 - 2ab + b^2\))
- Refined by Amirbekyan et al. in 2007 (corrected math)
- Implemented with practical extensions in GNUnet (negative numbers, small numbers, concrete protocol, set intersection, implementation).
Preliminaries

- Alice has public key $A$ and input map $m_A : M_A \rightarrow \mathbb{Z}$.
- Bob has public key $B$ and input map $m_B : M_B \rightarrow \mathbb{Z}$.
- We want to calculate
  \[ \sum_{i \in M_A \cap M_B} m_A(i)m_B(i) \] (15)
- We first calculate $M = M_A \cap M_B$.
- Define $a_i := m_A(i)$ and $b_i := m_B(i)$ for $i \in M$.
- Let $s$ denote a shared static offset.
Alice transmits $E_A(s + a_i)$ for $i \in M$ to Bob.

Bob creates two random permutations $\pi$ and $\pi'$ over the elements in $M$, and a random vector $r_i$ for $i \in M$ and sends

\begin{align*}
R : &= E_A(s + a_{\pi(i)}) \otimes E_A(s - r_{\pi(i)} - b_{\pi(i)}) \quad (16) \\
&= E_A(2 \cdot s + a_{\pi(i)} - r_{\pi(i)} - b_{\pi(i)}), \quad (17) \\
R' : &= E_A(s + a_{\pi'(i)}) \otimes E_A(s - r_{\pi'(i)}) \quad (18) \\
&= E_A(2 \cdot s + a_{\pi'(i)} - r_{\pi'(i)}), \quad (19) \\
S : &= \sum (r_i + b_i)^2, \quad (20) \\
S' : &= \sum r_i^2 \quad (21)
\end{align*}
Decryption (1/3)

Alice decrypts $R$ and $R'$ and computes for $i \in M$:

$$a_\pi(i) - b_\pi(i) - r_\pi(i) = D_A (R) - 2 \cdot s,$$

$$a_{\pi'}(i) - r_{\pi'}(i) = D_A (R') - 2 \cdot s,$$

which is used to calculate

$$T := \sum_{i \in M} a_i^2$$

$$U := -\sum_{i \in M} (a_\pi(i) - b_\pi(i) - r_\pi(i))^2$$

$$U' := -\sum_{i \in M} (a_{\pi'}(i) - r_{\pi'}(i))^2$$
Decryption (2/3)

She then computes

\[ P : = S + T + U \]

\[ = \sum_{i \in M} (b_i + r_i)^2 + \sum_{i \in M} a_i^2 + \left( - \sum_{i \in M} (a_i - b_i - r_i)^2 \right) \]

\[ = \sum_{i \in M} ((b_i + r_i)^2 + a_i^2 - (a_i - b_i - r_i)^2) \]

\[ = 2 \cdot \sum_{i \in M} a_i (b_i + r_i). \]

\[ P' : = S' + T + U' \]

\[ = \sum_{i \in M} r_i^2 + \sum_{i \in M} a_i^2 + \left( - \sum_{i \in M} (a_i - r_i)^2 \right) \]

\[ = \sum_{i \in M} (r_i^2 + a_i^2 - (a_i - r_i)^2) = 2 \cdot \sum_{i \in M} a_i r_i. \]
Finally, Alice computes the scalar product using:

\[
\frac{P - P'}{2} = \sum_{i \in M} a_i (b_i + r_i) - \sum_{i \in M} a_i r_i = \sum_{i \in M} a_i b_i. \quad (27)
\]
Who said calculating DLOG was hard?
Alice’s public key ist \( A = g^a \), ihr private key ist \( a \). Alices schickt an Bob \((g_i, h_i) = (g^{r_i}, g^{r_ia+a_i})\) mit zufälligen Werten \( r_i \) für \( i \in M \).

Bob antwortet mit

\[
\left( \prod_{i \in M} g_i^{b_i}, \prod_{i \in M} h_i^{b_i} \right) = \left( \prod_{i \in M} g_i^{b_i}, (\prod_{i \in M} g_i^{b_i})^a g \sum_{i \in M} a_i b_i \right)
\]

Alice kann dann berechnen

\[
\left( \prod_{i \in M} g_i^{b_i} \right)^{-a} \cdot \left( \prod_{i \in M} g_i^{b_i} \right)^a \cdot g \sum_{i \in M} a_i b_i = g \sum_{i \in M} a_i b_i.
\]

Falls \( \sum_{i \in M} a_i b_i \) ausreichend klein ist, kann Alice dann das Skalarprodukt durch Lösung des DLP bestimmen.

---

1 Joint work with Tanja Lange
## Performance Evaluation

<table>
<thead>
<tr>
<th>Length</th>
<th>RSA-2048</th>
<th>ECC-2^{20}</th>
<th>ECC-2^{28}</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>14 s</td>
<td>2 s</td>
<td>29 s</td>
</tr>
<tr>
<td>50</td>
<td>21 s</td>
<td>2 s</td>
<td>29 s</td>
</tr>
<tr>
<td>100</td>
<td>39 s</td>
<td>2 s</td>
<td>29 s</td>
</tr>
<tr>
<td>200</td>
<td>77 s</td>
<td>3 s</td>
<td>30 s</td>
</tr>
<tr>
<td>400</td>
<td>149 s</td>
<td>OOR</td>
<td>31 s</td>
</tr>
<tr>
<td>800</td>
<td>304 s</td>
<td>OOR</td>
<td>33 s</td>
</tr>
<tr>
<td>800</td>
<td>3846 kb</td>
<td>OOR</td>
<td>70 kb</td>
</tr>
</tbody>
</table>

The pre-calculation of ECC-2^{28} is $\times 16$ more expensive than for ECC-2^{20} as the table is set to have size $\sqrt{n}$.  

Exercise

Implement function to calculate DLOG.