

# Non-boring cryptography

Christian Grothoff

Berner Fachhochschule

5.6.2020

# Learning Objectives

- ▶ Homomorphic encryption
- ▶ Secure multiparty computation (SMC) theory
- ▶ Common SMC adversary models
- ▶ Specialized protocols for SMC problems
- ▶ Fundamental design limitations for Blockchains
- ▶ Blind signatures and applications to payments
- ▶ Key management issues and solutions

# 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

$$E(x_1 \oplus x_2) = E(x_1) \otimes E(x_2) \quad (1)$$

# Multiplicative Homomorphism: RSA & ElGamal

- ▶ Unpadded RSA (multiplicative):

$$E(x_1) \cdot E(x_2) = x_1^e x_2^e = E(x_1 \cdot x_2) \quad (2)$$

- ▶ ElGamal:

$$E(x_1) \cdot E(x_2) = (g^{r_1}, x_1 \cdot h^{r_1})(g^{r_2}, x_2 \cdot h^{r_2}) \quad (3)$$

$$= (g^{r_1+r_2}, (x_1 \cdot x_2)h^{r_1+r_2}) \quad (4)$$

$$= E(m_1 \cdot m_2) \quad (5)$$

## Additive Homomorphism: Paillier

$$E_K(m) := g^m \cdot r^n \pmod{n^2}, \quad (6)$$

$$D_K(c) := \frac{(c^\lambda \pmod{n^2}) - 1}{n} \cdot \mu \pmod{n} \quad (7)$$

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:

$$\lambda := \text{lcm}(p-1, q-1), \quad (8)$$

$$\mu := \left( \frac{(g^\lambda \pmod{n^2}) - 1}{n} \right)^{-1} \pmod{n}. \quad (9)$$

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

$$E_K(a) \otimes E_K(b) \equiv E_K(a+b) \quad (10)$$

for any public key  $K$ .

# Fully homomorphic encryption

Additive:

$$E(A) \oplus E(B) = E(A + B) \quad (11)$$

**and** multiplicative:

$$E(A) \otimes E(B) = E(A \cdot B) \quad (12)$$

Known cryptosystems: Brakerski-Gentry-Vaikuntanathan (BGV), NTRU, Gentry-Sahai-Waters (GSW).

**Break**

## Example: Secure Scalar Product

- ▶ Original idea by Ioannidis et al. in 2002 (use:  $(a - b)^2 = a^2 - 2ab + b^2$ )
- ▶ Refined by Amirbekyan et al. in 2007 (corrected math)
- ▶ Now providing protocol with practical extensions (negative numbers, small numbers, set intersection).

## 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

$$\sum_{i \in M_A \cap M_B} m_A(i)m_B(i) \tag{13}$$

- ▶ 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.

# Network Protocol

- ▶ 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

$$R := E_A(s + a_{\pi(i)}) \otimes E_A(s - r_{\pi(i)} - b_{\pi(i)}) \quad (14)$$

$$= E_A(2 \cdot s + a_{\pi(i)} - r_{\pi(i)} - b_{\pi(i)}), \quad (15)$$

$$R' := E_A(s + a_{\pi'(i)}) \otimes E_A(s - r_{\pi'(i)}) \quad (16)$$

$$= E_A(2 \cdot s + a_{\pi'(i)} - r_{\pi'(i)}), \quad (17)$$

$$S := \sum (r_i + b_i)^2, \quad (18)$$

$$S' := \sum r_i^2 \quad (19)$$



## Decryption (1/3)

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

$$a_{\pi(i)} - b_{\pi(i)} - r_{\pi(i)} = D_A(R) - 2 \cdot s, \quad (20)$$

$$a_{\pi'(i)} - r_{\pi'(i)} = D_A(R') - 2 \cdot s, \quad (21)$$

which is used to calculate

$$T := \sum_{i \in M} a_i^2 \quad (22)$$

$$U := - \sum_{i \in M} (a_{\pi(i)} - b_{\pi(i)} - r_{\pi(i)})^2 \quad (23)$$

$$U' := - \sum_{i \in M} (a_{\pi'(i)} - r_{\pi'(i)})^2 \quad (24)$$

## Decryption (2/3)

She then computes

$$\begin{aligned}P &:= S + T + U \\&= \sum_{i \in M} (b_i + r_i)^2 + \sum_{i \in M} a_i^2 + \left( - \sum_{i \in M} (a_i - b_i - r_i)^2 \right) \\&= \sum_{i \in M} ((b_i + r_i)^2 + a_i^2 - (a_i - b_i - r_i)^2) \\&= 2 \cdot \sum_{i \in M} a_i (b_i + r_i).\end{aligned}$$

$$\begin{aligned}P' &:= S' + T + U' \\&= \sum_{i \in M} r_i^2 + \sum_{i \in M} a_i^2 + \left( - \sum_{i \in M} (a_i - r_i)^2 \right) \\&= \sum_{i \in M} (r_i^2 + a_i^2 - (a_i - r_i)^2) = 2 \cdot \sum_{i \in M} a_i r_i.\end{aligned}$$

## Decryption (3/3)

Finally, Alice computes the scalar product using:

$$\frac{P - P'}{2} = \sum_{i \in M} a_i(b_i + r_i) - \sum_{i \in M} a_i r_i = \sum_{i \in M} a_i b_i. \quad (25)$$

# Computing Discrete Logarithms

Who said calculating DLOG was hard?

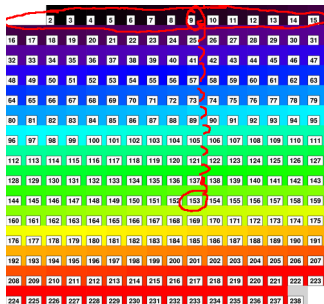
## Computing Discrete Logarithms

		2	3	4	5	6	7	8	9	10
16	17	18	19	20	21	22	23	24	25	26
32	33	34	35	36	37	38	39	40	41	42
48	49	50	51	52	53	54	55	56	57	58
64	65	66	67	68	69	70	71	72	73	74
80	81	82	83	84	85	86	87	88	89	90
96	97	98	99	100	101	102	103	104	105	106

# Baby Steps



# Giant Steps



Alice's public key ist  $A = g^a$ , ihr private key ist  $a$ . Alices schickt an Bob  $(g_i, h_i) = (g^{r_i}, g^{r_i a + a_i})$  mit zufälligen Werten  $r_i$  für  $i \in M$ . Bob antwortet mit

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

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.

---

<sup>1</sup>Joint work with Tanja Lange



## Performance Evaluation

Length	RSA-2048	ECC-2 <sup>20</sup>	ECC-2 <sup>28</sup>
25	14 s	2 s	29 s
50	21 s	2 s	29 s
100	39 s	2 s	29 s
200	77 s	3 s	30 s
400	149 s	OOB	31 s
800	304 s	OOB	33 s
800	3846 kb	OOB	70 kb

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

# Homework

Implement function to calculate DLOG.

**Break**

## Reminder: RSA

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

$$\text{GCD}((p-1)(q-1), e) = 1 \quad (26)$$

- ▶ Define  $n = pq$ ,
- ▶ compute  $d$  such that  $ed \equiv 1 \pmod{(p-1)(q-1)}$ .
- ▶ Let  $s := m^d \pmod{n}$ .
- ▶ Then  $m \equiv s^e \pmod{n}$ .

# RSA Summary

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

# Low Encryption Exponent Attack

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

$$D_{A_{priv}}(D_{B_{priv}}(E_{A_{pub}}(E_{B_{pub}}(M)))) \neq M \quad (27)$$

- ▶ PKCS#1 / RFC 3447 define a padding standard

# Blind signatures with RSA

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



# Blind signatures with RSA

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

# Blind signatures with RSA

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

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

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

4. Transmit  
 $f' := fb^e \pmod n$

1. Receive  $f'.$

2. Compute  
 $s' := f'^d \pmod n.$

3. Send  $s'.$

1. Receive  $s'.$

2. Compute  
 $s := s'b^{-1} \pmod n$

**What domain of digital communication should we be most concerned about?**

# Surveillance concerns

- ▶ Everybody knows about Internet surveillance.
- ▶ But is it **that** bad?

# Surveillance concerns

- ▶ Everybody knows about Internet surveillance.
- ▶ But is it **that** bad?
  - ▶ You can choose when and where to use the Internet
  - ▶ You can anonymously access the Web using Tor
  - ▶ You can find open access points that do not require authentication
  - ▶ IP packets do not include your precise location or name
  - ▶ ISPs typically store this meta data for days, weeks or months

## Where is it worse?

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?”*

## Where is it worse?

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?”*



## What is worse:

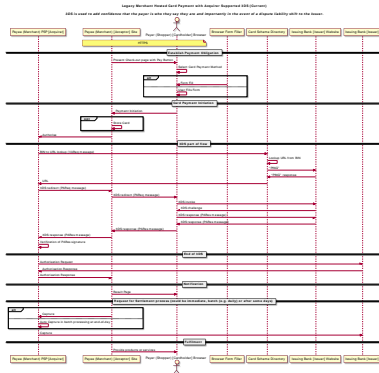
- ▶ When you pay by CC, the information includes your name
- ▶ When you pay in person with CC, your location is also known
- ▶ You often have no alternative payment methods available
- ▶ You hardly ever can use someone else's CC
- ▶ Anonymous prepaid cards are difficult to get and expensive
- ▶ Payment information is typically stored for at least 6 years



# Banks have Problems, too!

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



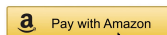
Online credit card payments will be replaced, but with what?

# The Bank's Problem

- ▶ Global tech companies push oligopolies
- ▶ Privacy and federated finance are at risk
- ▶ Economic sovereignty is in danger

**PayPal**<sup>™</sup>

支付宝<sup>™</sup>  
Alipay.com



Apple Pay



## Predicting the Future

- ▶ Google and Apple will be your bank and run your payment system
- ▶ They can target advertising based on your purchase history, location and your ability to pay
- ▶ They will provide more usable, faster and broadly available payment solutions; our federated banking system will be history
- ▶ After they dominate the payment sector, they will start to charge fees befitting their oligopoly size
- ▶ Competitors and vendors not aligning with their corporate “values” will be excluded by policy and go bankrupt
- ▶ The imperium will have another major tool for its financial warfare

**Do you want to live under total surveillance?**

# Banking, Surveillance and Physical Security

<https://www.youtube.com/watch?v=GyJZViNf2Vk> (4'2019)

**Break**

# Bitcoin

Payment system using a block chain:

- ▶ Public keys identify accounts, private keys used to send money from the account into other accounts.
- ▶ Set of internally consistent transactions form block
- ▶ Each block includes a transaction creating fresh coins and transferring applicable fees to block creator
- ▶ Difficulty adjusts to mining power to mine a block in  $\approx 10$  minutes
- ▶ Amount of coins created per block is exponentially decreasing

# Mining

Mining requires:

- ▶ Learning pending transactions from peers
- ▶ Selecting a subset of of transactions which is valid (no double spending) by computing current account balances against the entire history
- ▶ Finding a hash collision (with adaptive difficulty)
- ▶ Propagating the new block to other miners

Usually specialized systems are used for finding hash collisions.



# The CAP Theorem

We like three properties in distributed systems:

- ▶ Consistency: everybody agrees on the state
- ▶ Availability: the system is always available
- ▶ Partition tolerance: the system handles component (host, link) failure, including network partitions

CAP Theorem is an **impossibility proof**: you can only have 2/3.

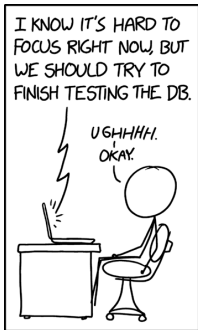
# CAP & Bitcoin

Bitcoin is inconsistent:

- ▶ Conflicting blocks can be mined at the same time
- ▶ This can happen by accident, or on purpose!
- ▶ Coins could be spent twice, once on each fork of the chain!
- ▶ Longest chain is considered “valid”
- ▶ Original paper suggests to consider transaction confirmed only after at least 6 blocks past the transaction.

⇒ Bitcoin is not consistent.

⇒ Competitively long alternative chains void durability even after 6 blocks!



THE SYSTEM NEEDS TO GUARANTEE EVENTUAL CONSISTENCY.



EVENTUAL CONSISTENCY IS GUARANTEED BY THE 2ND LAW OF THERMODYNAMICS.

SOONER OR LATER THIS WILL ALL BE A UNIFORM HEAT BATH.

MAXIMUM ENTROPY.



MAXIMUM ENTROPY MEANS NO USEFUL WORK CAN BE DONE!

I'M GETTING A HEAD START BY DOING NO USEFUL WORK NOW!



<https://xkcd.com/2315/>

# Bitcoin performance

- ▶ Privacy: all transactions happen in the clear in public view
- ▶ Latency: transactions take 1h to kind-of be confirmed
- ▶ Storage: grows linearly forever, no garbage collection
- ▶ Power: mining consumes more than the entire state of Denmark today
- ▶ Rate: Network handles at most about 7 transactions per second
- ▶ Accountability: use of public keys as addresses enables criminal use

⇒ Bitcoin fever lasting for years. Why?

# Altcoins

- ▶ Dogecoin: same as Bitcoin, just named after a dog meme (an idea that is obviously worth billions!)
- ▶ Zcash: uses ZKSNARKs<sup>2</sup> to hide transactions (criminal activity on Bitcoin was too low)
- ▶ Ethereum: run Turing-complete virtual machine logic in the blockchain to enable “smart” contracts and arbitrary applications, not just payments (is “Accelerando” an utopia or dystopia?)

Experimental designs promising to drastically improve performance (Bolt, Lightning) have so far failed to deliver.

---

<sup>2</sup>≈ 1-15 minutes CPU time to create new transaction needed!

## Case study: Private payments

*“A company is developing new software for private payments. This will enable its customers to transact with “complete” privacy (like cash). The solution does not include backdoors, and thus the company cannot block payments to support trade embargos or anti money laundering efforts.”*

- ▶ Discuss virtues and vices affected.
- ▶ Does it make a difference if the software is Free Software developed by a community instead of proprietary software from a company?
- ▶ Suppose the company added a feature to provide income transparency where the state gets to see who receives funds (but not who made payments). Does this change your assessment?

**Break**

**Digital** cash, made **socially**  
**responsible.**

**< T a l e r >**

Privacy-Preserving, Practical, Taxable, Free Software, Efficient



# What is Taler?

Taler is an electronic instant payment system.

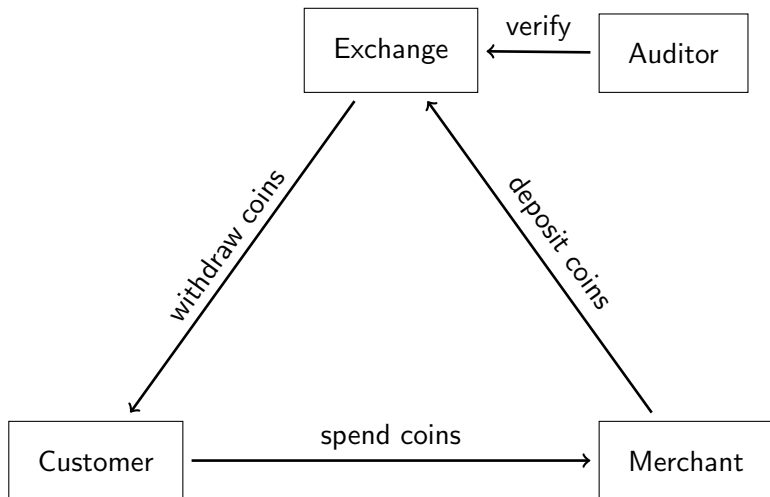
- ▶ Uses electronic coins stored in **wallets** on customer's device
- ▶ Like **cash**
- ▶ Pay in **existing currencies** (i.e. EUR, USD, BTC),  
or use it to create new **regional currencies**

# Design goals for the GNU Taler Payment System

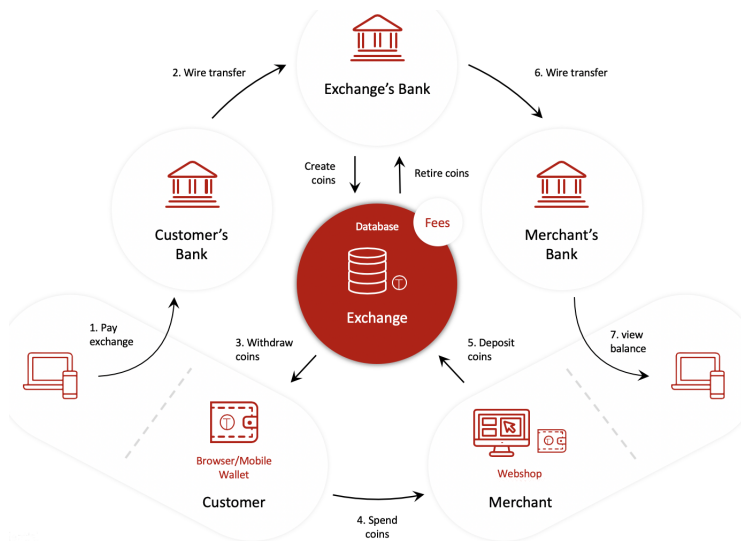
GNU Taler must ...

1. ... be implemented as **free software**.
2. ... protect the **privacy of buyers**.
3. ... must enable the state to **tax income** and crack down on illegal business activities.
4. ... prevent payment fraud.
5. ... only **disclose the minimal amount of information necessary**.
6. ... be usable.
7. ... be efficient.
8. ... avoid single points of failure.
9. ... foster **competition**.

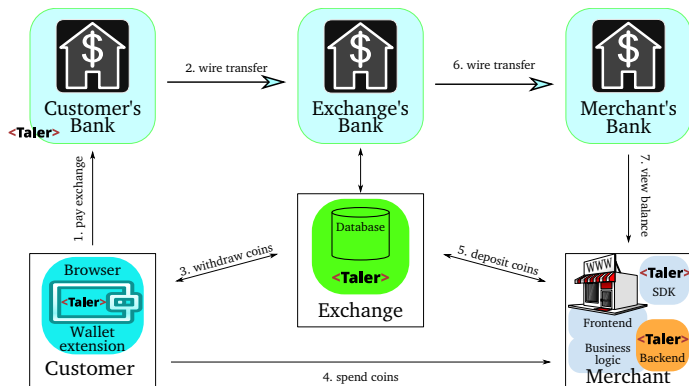
# Taler Overview



# Architecture of Taler



# Architecture of Taler



⇒ Convenient, taxable, privacy-enhancing, & resource friendly!

# Usability of Taler

`https://demo.taler.net/`

1. Install Web extension.
2. Visit the `bank.demo.taler.net` to withdraw coins.
3. Visit the `shop.demo.taler.net` to spend coins.

# Taxability

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.

# Taxability

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?

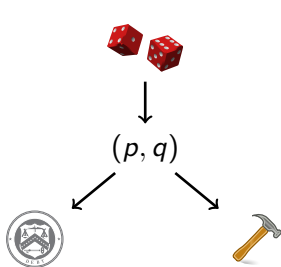
We use a few ancient constructions:

- ▶ 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  $d := e^{-1} \pmod{\phi(n)}$   
exists.
4. Publish public key  $(e, n)$ .



# Merchant: Create a signing key (EdDSA)

- ▶ pick random  $m \pmod{o}$  as private key
- ▶  $M = mG$  public key



↓  
 $m$

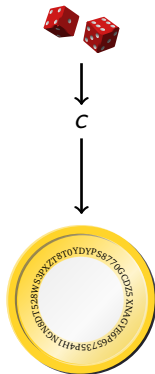
↓  
 $M$

**Capability:**  $m \Rightarrow$



# Customer: Create a planchet (EdDSA)

- ▶ Pick random  $c \pmod{o}$  private key
- ▶  $C = cG$  public key

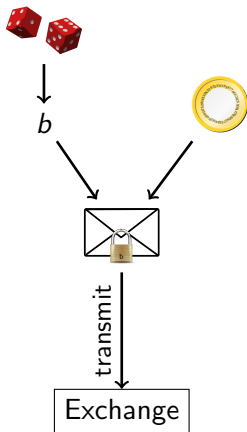


**Capability:**  $c \Rightarrow$



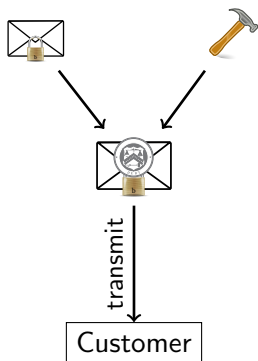
## Customer: Blind planchet (RSA)

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



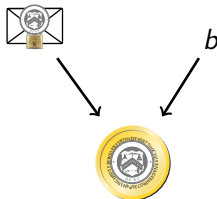
## Exchange: Blind sign (RSA)

1. Receive  $f'$ .
2. Compute  $s' := f'^d \pmod n$ .
3. Send signature  $s'$ .

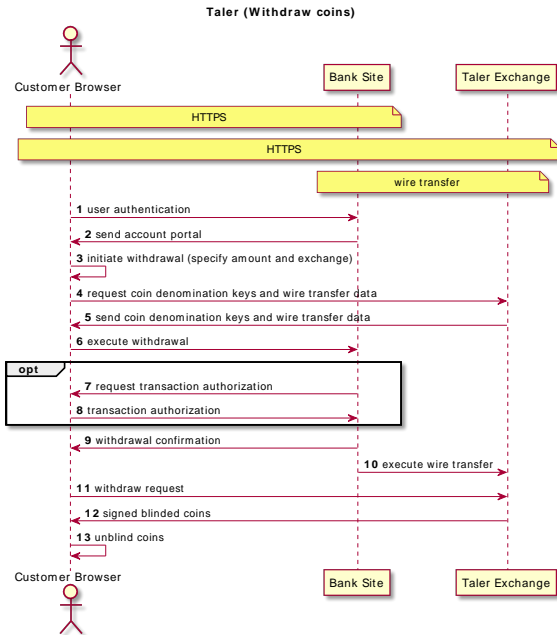


## Customer: Unblind coin (RSA)

1. Receive  $s'$ .
2. Compute  $s := s'b^{-1} \bmod n$

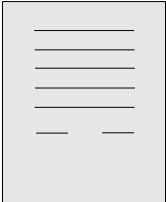


# Withdrawing coins on the Web





# Customer: Build shopping cart



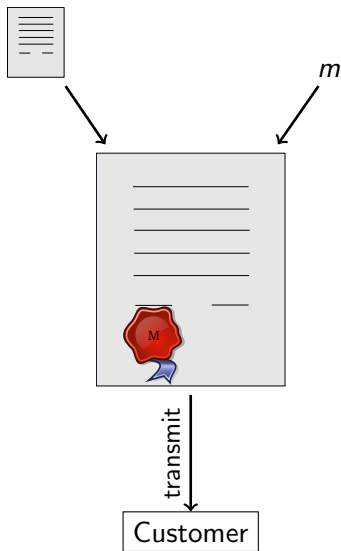
transmit



Merchant

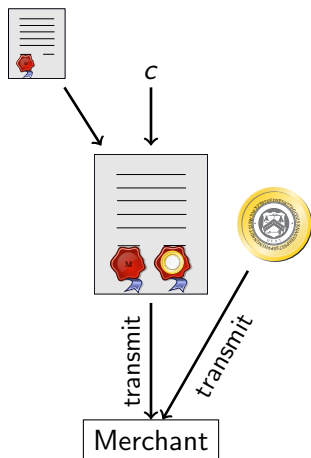
## Merchant: Propose contract (EdDSA)

1. Complete proposal  $D$ .
2. Send  $D, EdDSA_m(D)$



## Customer: Spend coin (EdDSA)

1. Receive proposal  $D$ ,  $EdDSA_m(D)$ .
2. Send  $s$ ,  $C$ ,  $EdDSA_c(D)$

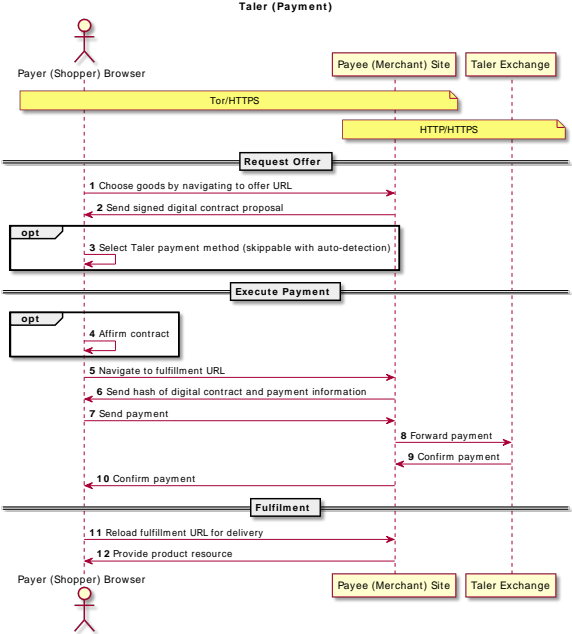


# Merchant and Exchange: Verify coin (RSA)

$$s^e \stackrel{?}{\equiv} m \pmod{n}$$



# Payment processing with Taler



## Warranting deposit safety

Exchange has *another* online signing key  $W = wG$ :

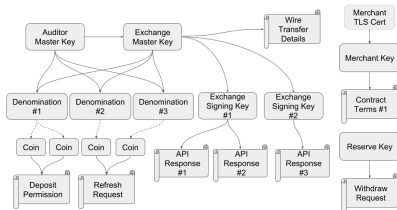
Sends  $E, EdDSA_w(M, H(D), FDH(C))$  to the merchant.

This signature means that  $M$  was the *first* to deposit  $C$  and that the exchange thus must pay  $M$ .

Without this, an evil exchange could renege on the deposit confirmation and claim double-spending if a coin were deposited twice, and then not pay either merchant!

# Online keys

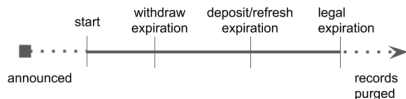
- ▶ The exchange needs  $d$  and  $w$  to be available for online signing.
- ▶ The corresponding public keys  $W$  and  $(e, n)$  are certified using Taler's public key infrastructure (which uses offline-only keys).



**What happens if those private keys are compromised?**

## Denomination key ( $e, n$ ) compromise

- ▶ An attacker who learns  $d$  can sign an arbitrary number of illicit coins into existence and deposit them.
  - ▶ Auditor and exchange can detect this once the total number of deposits (illicit and legitimate) exceeds the number of legitimate coins the exchange created.
  - ▶ At this point,  $(e, n)$  is *revoked*. Users of *unspent* legitimate coins reveal  $b$  from their withdrawal operation and obtain a *refund*.
  - ▶ The financial loss of the exchange is *bounded* by the number of legitimate coins signed with  $d$ .
- ⇒ Taler frequently rotates denomination signing keys and deletes  $d$  after the signing period of the respective key expires.





## Online signing key $w$ compromise

- ▶ An attacker who learns  $w$  can sign deposit confirmations.
- ▶ Attacker sets up two (or more) merchants and customer(s) which double-spend legitimate coins at both merchants.
- ▶ The merchants only deposit each coin once at the exchange and get paid once.
- ▶ The attacker then uses  $w$  to fake deposit confirmations for the double-spent transactions.
- ▶ The attacker uses the faked deposit confirmations to complain to the auditor that the exchange did not honor the (faked) deposit confirmations.

The auditor can then detect the double-spending, but cannot tell who is to blame, and (likely) would presume an evil exchange, forcing it to pay both merchants.

**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:

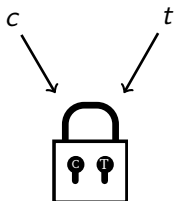
- ▶ maintain unlinkability
- ▶ maintain taxability of transactions

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)

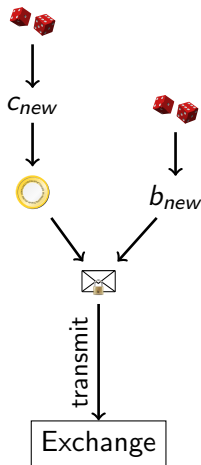
1. Create private keys  $c, t$   
mod  $o$
2. Define  $C = cG$
3. Define  $T = tG$
4. Compute DH  
 $cT = c(tG) = t(cG) =$   
 $tC$



# Strawman solution

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

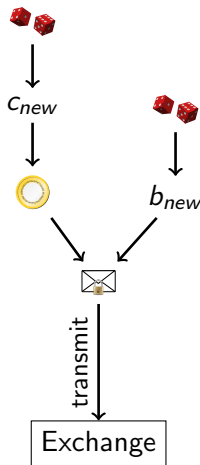
1. Pick random  $c_{new}$  mod  $o$  private key
  2.  $C_{new} = c_{new}G$  public key
  3. Pick random  $b_{new}$
  4. Compute  $f_{new} := FDH(C_{new})$ ,  $m < n$ .
  5. Transmit  $f'_{new} := f_{new}b_{new}^e \pmod n$
- ... and sign request for change with  $c_{old}$ .



# 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  $c_{old}$ .



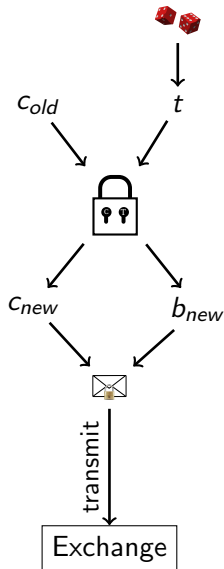
**Problem: Owner of  $C_{new}$  may differ from owner of  $C_{old}$ !**



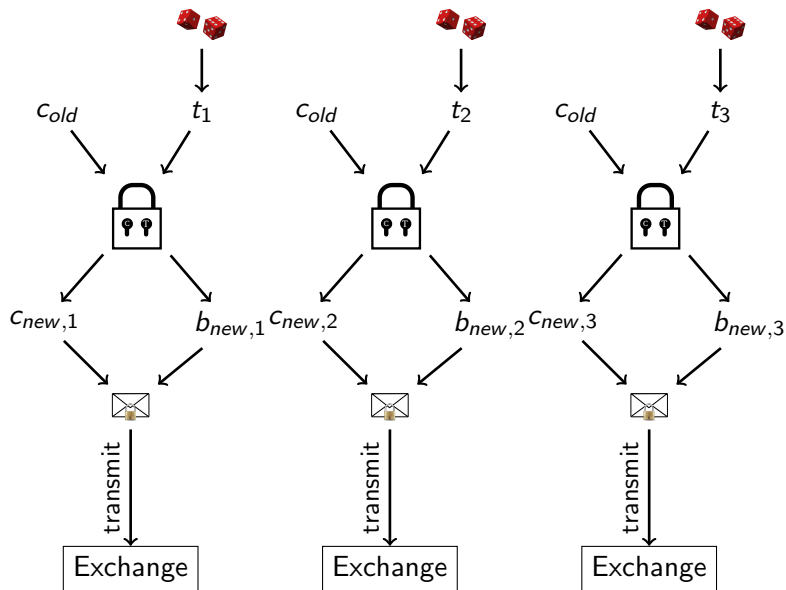
# Customer: Transfer key setup (ECDH)

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

1. Let  $C_{old} := c_{old} G$  (as before)
2. Create random private transfer key  $t \pmod{o}$
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



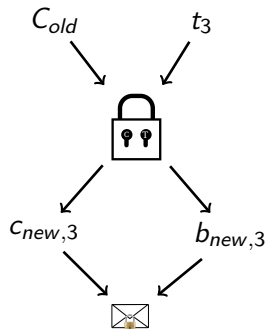
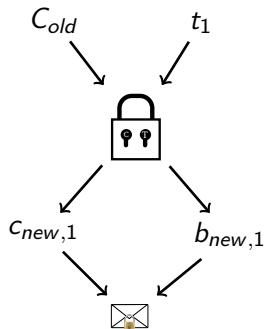
Exchange: Choose!

Exchange sends back random  $\gamma \in \{1, 2, 3\}$  to the customer.

## Customer: Reveal

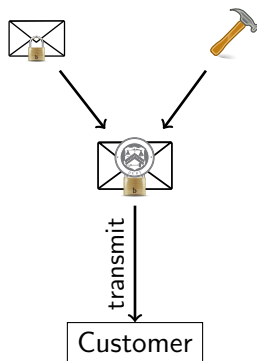
1. If  $\gamma = 1$ , send  $t_2, t_3$  to exchange
2. If  $\gamma = 2$ , send  $t_1, t_3$  to exchange
3. If  $\gamma = 3$ , send  $t_1, t_2$  to exchange

## Exchange: Verify ( $\gamma = 2$ )



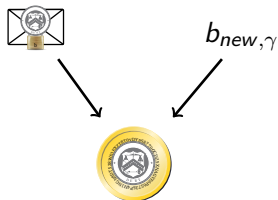
## Exchange: Blind sign change (RSA)

1. Take  $f'_{new,\gamma}$ .
2. Compute  $s' := f'^d_{new,\gamma} \bmod n$ .
3. Send signature  $s'$ .



## Customer: Unblind change (RSA)

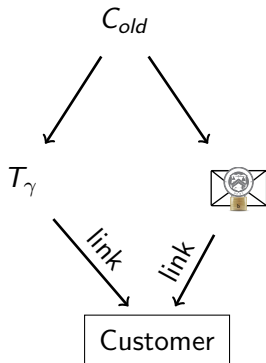
1. Receive  $s'$ .
2. Compute  $s := s' b_{new,\gamma}^{-1} \pmod n$ .



## Exchange: Allow linking change

Given  $C_{old}$

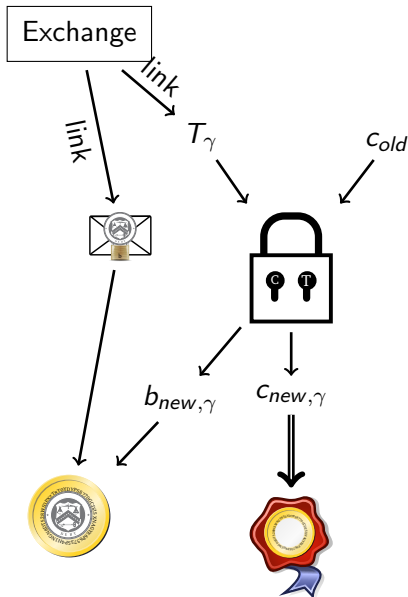
return  $T_\gamma$  and

$$s := s' b_{new,\gamma}^{-1} \pmod n.$$




# 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  $s := s' b_{new,\gamma}^{-1} \pmod n$



## 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:

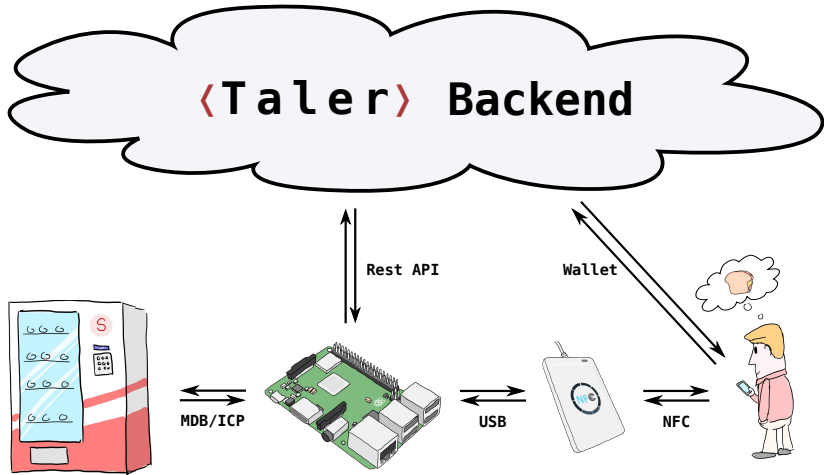
- ▶ To give unlinkable change.
- ▶ To give refunds to an anonymous customer.
- ▶ To expire old keys and migrate coins to new ones.
- ▶ To handle protocol aborts.

**Transactions via refresh are equivalent to sharing a wallet.**

# Example: The Taler Snack Machine<sup>3</sup>

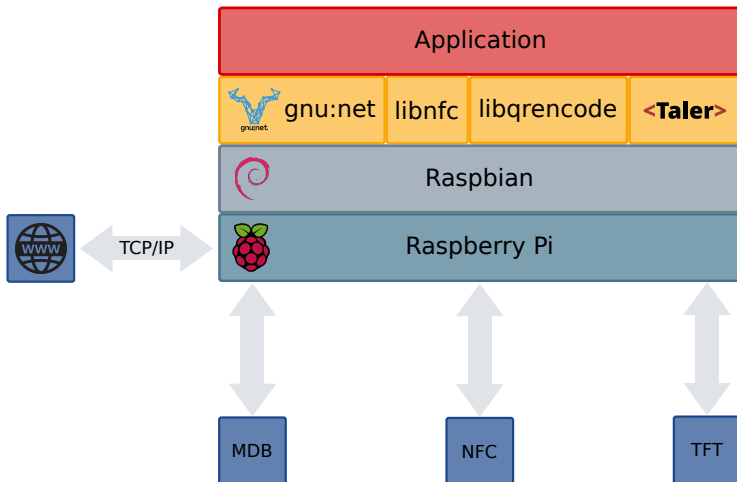
Integration of a MDB/ICP to Taler gateway.

Implementation of a NFC or QR-Code to Taler wallet interface.

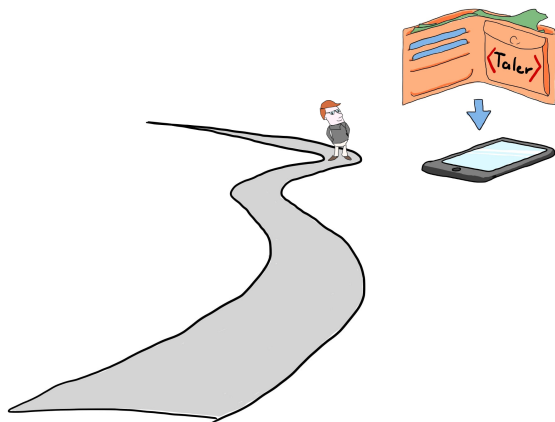


<sup>3</sup>By M. Boss and D. Hofer

# Software architecture for the Taler Snack Machine



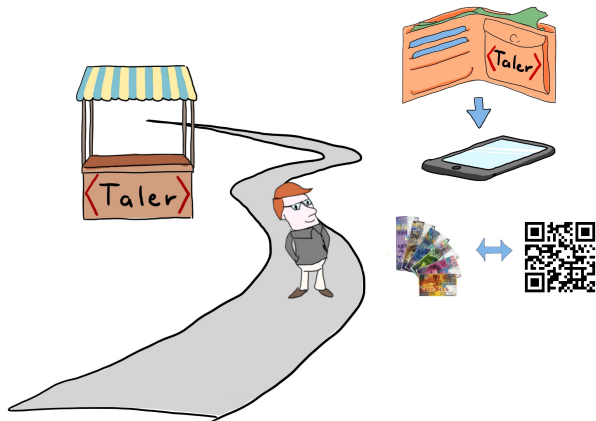
## User story: Install App on Android<sup>4</sup>



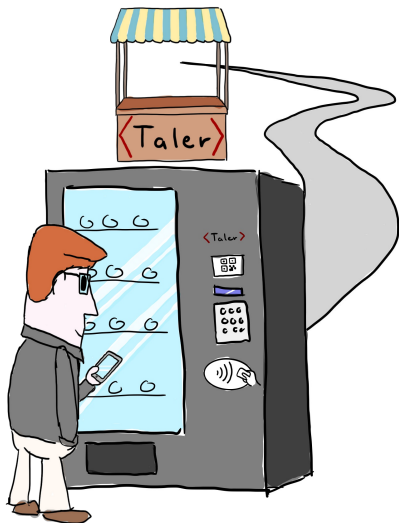
---

<sup>4</sup><https://wallet.taler.net/>

## User story: Withdraw e-cash

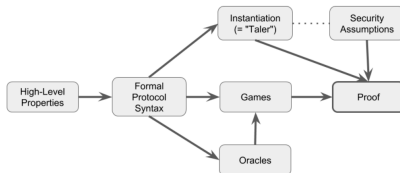


User story: Use machine!



# Summary

- ▶ We can design protocols that fail *soft*.
- ▶ GNU Taler's design limits financial damage even in the case private keys are compromised.
- ▶ GNU Taler does:
  - ▶ Gives change, can provide refunds
  - ▶ Integrates nicely with HTTP, handles network failures
  - ▶ High performance
  - ▶ Free Software
  - ▶ Formal security proofs





**Break**

## **Anastasis<sup>5</sup>**

---

<sup>5</sup>Based on a BFH Bachelor's thesis by D. Neufeld and D. Meister

# THE PROBLEM



Confidentiality requires only consumer is in control of key material



Consumers are unable to simultaneously ensure confidentiality and availability of keys



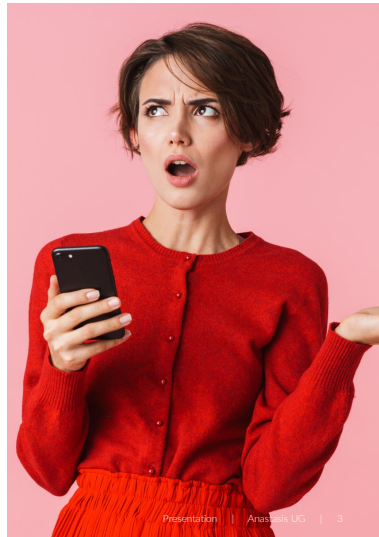
Cryptographic key-splitting solutions so far are not usable



European e-money issuers using electronic wallets must:<sup>1</sup>

- Enable consumers to always recover their electronic funds (i.e. if devices are lost)
- Not assume consumers are able to remember or securely preserve key material

<sup>1</sup> According to communication from ECB to Taler Systems SA.



# THE PROBLEM



Confidentiality requires only consumer is in control of key material



Consumers are unable to simultaneously ensure confidentiality and availability of keys



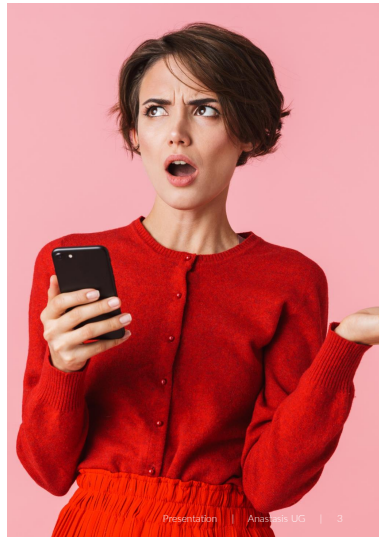
Cryptographic key-splitting solutions so far are not usable



European e-money issuers using electronic wallets must:<sup>1</sup>

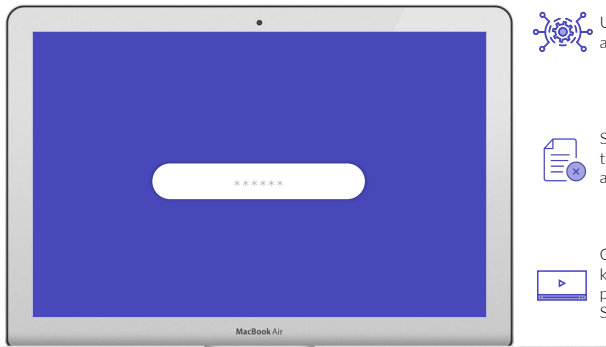
- Enable consumers to always recover their electronic funds (i.e. if devices are lost)
- Not assume consumers are able to remember or securely preserve key material

<sup>1</sup> According to communication from ECB to Taler Systems SA.



# WHAT IS ANASTASIS?

**ANASTASIS IS A KEY RECOVERY SERVICE.**



Users split their secret keys across multiple service providers

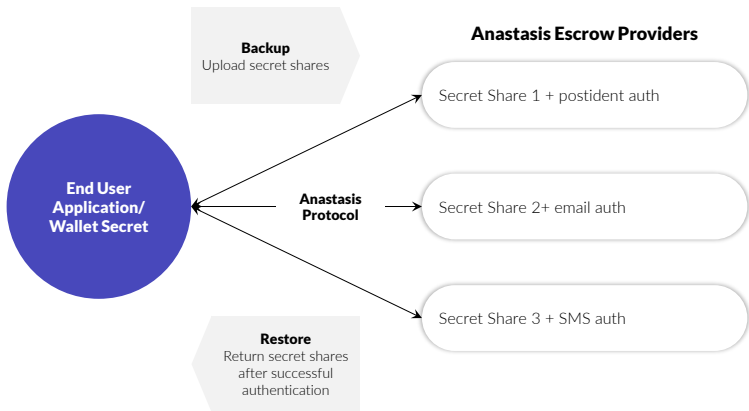


Service providers learn nothing about the user, except possibly some details about how to authenticate the user



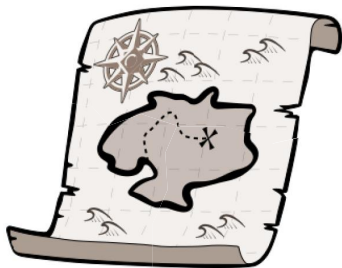
Only the authorized user can recover the key by following standard authentication procedures (SMS TAN, Video-Ident, Security Question, eMail, etc.)

# OVERVIEW



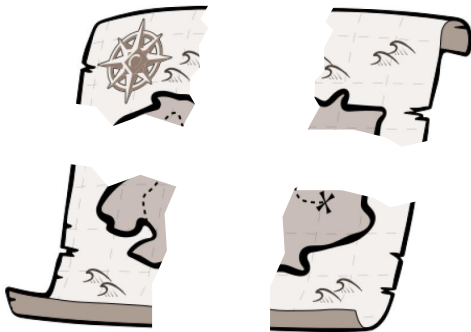
# SIMPLIFIED PROCESS FLOW

## STEP 1: RECOVERY INFORMATION



# SIMPLIFIED PROCESS FLOW

## STEP 2: SPLIT RECOVERY INFORMATION





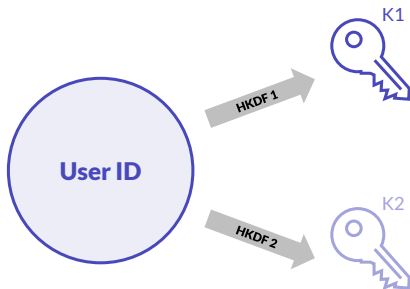
# SIMPLIFIED PROCESS FLOW

## STEP 3: USER IDENTIFICATION



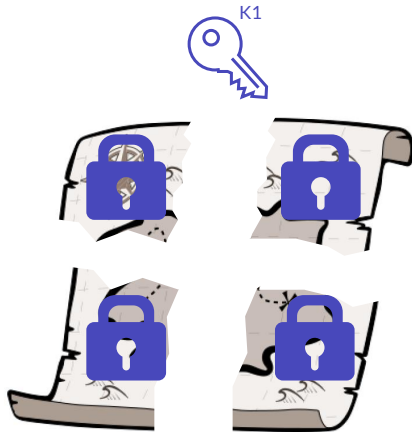
# SIMPLIFIED PROCESS FLOW

## STEP 4: KEY DERIVATION



# SIMPLIFIED PROCESS FLOW

## STEP 5: ENCRYPT PARTS



# SIMPLIFIED PROCESS FLOW

STEP 6: ADD TRUTH



+ H (answer to  
security question)



+ Picture



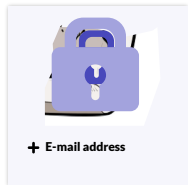
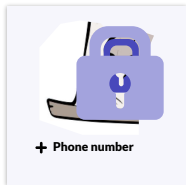
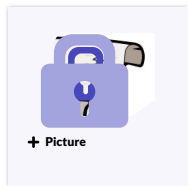
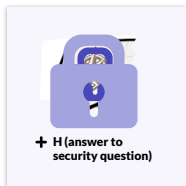
+ Phone number



+ E-mail address

# SIMPLIFIED PROCESS FLOW

STEP 7: ENCRYPT  
TRUTH



# SIMPLIFIED PROCESS FLOW

STEP 8: STORE  
DATA



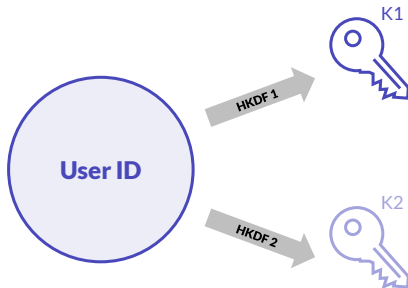
# SIMPLIFIED PROCESS FLOW

## STEP 9: USER IDENTIFICATION



# SIMPLIFIED PROCESS FLOW

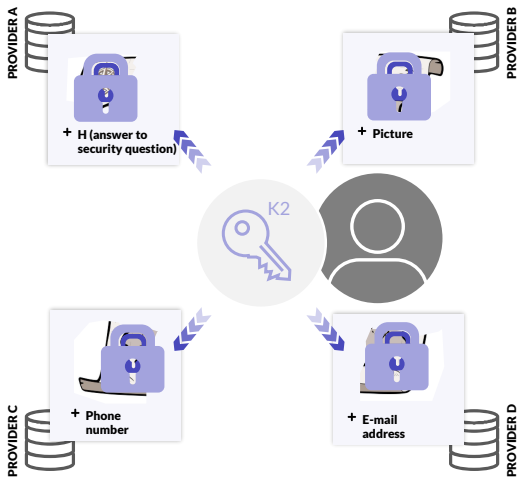
## STEP 10: KEY DERIVATION





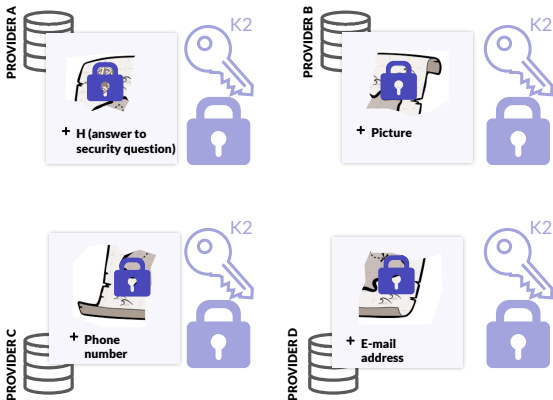
# SIMPLIFIED PROCESS FLOW

STEP 11:  
PROVIDE KEY



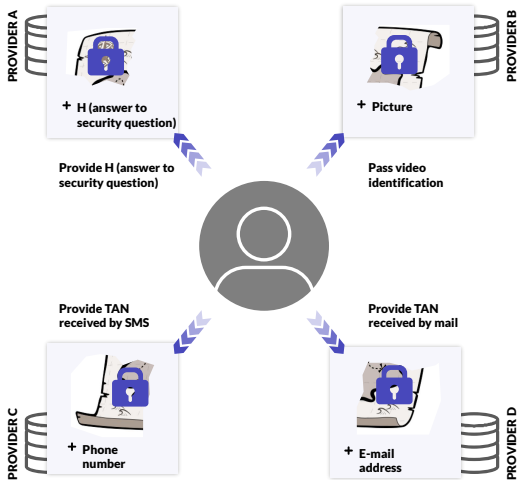
# SIMPLIFIED PROCESS FLOW

STEP 12:  
DECRYPT TRUTH



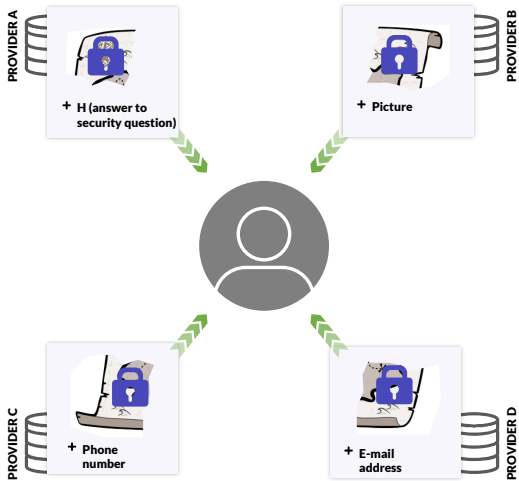
# SIMPLIFIED PROCESS FLOW

## STEP 13: AUTHENTICATION



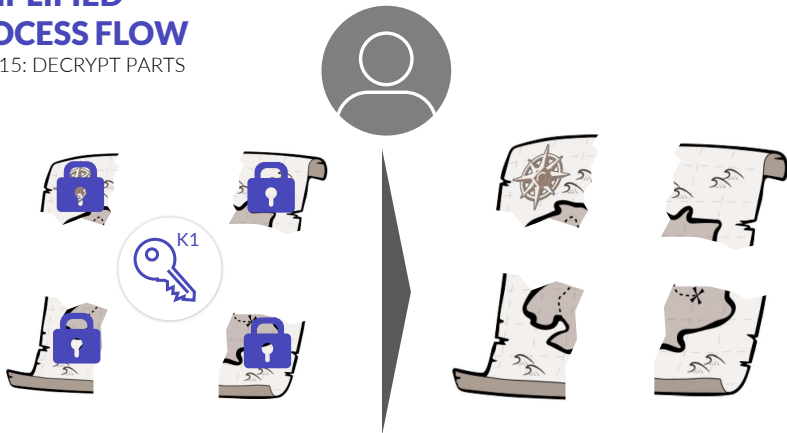
# SIMPLIFIED PROCESS FLOW

STEP 14:  
RECEIVE PARTS



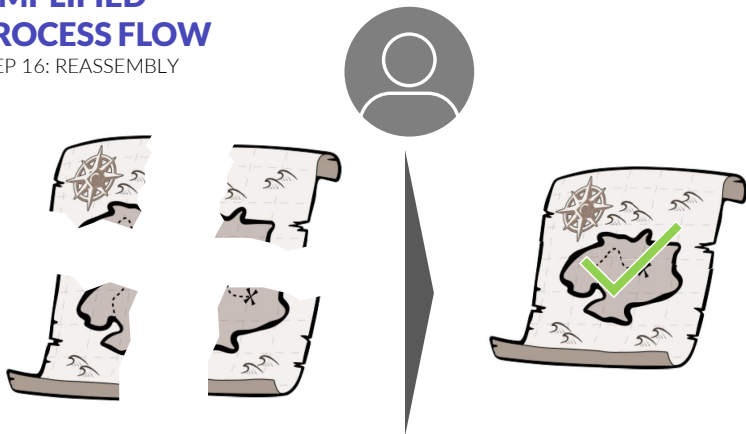
# SIMPLIFIED PROCESS FLOW

STEP 15: DECRYPT PARTS



# SIMPLIFIED PROCESS FLOW

STEP 16: REASSEMBLY



# SIMPLIFICATIONS

THE PREVIOUS ILLUSTRATION MAKES VARIOUS SIMPLIFICATIONS



Policies to allow  
more flexible  
splitting than 4/4



Recovery document  
to remember policies  
and providers



Distinction between  
core secret and  
master secret



Payment  
processing



Provider  
salts



Anti-DDoS provisions  
in protocol /  
request limits



Versioning



Liability  
limitations

# UNIQUE SALES PROPOSITIONS (USPS)

**1** Distributed trust instead of single point of failure

**4** Ease of use

**7** Generic API suitable for a range of applications

**2** Maximum privacy with respect to authentication data

**5** Low cost, scalable cloud-based solution

**8** Customers can remain anonymous:

- Minimizes risk to Anastasis service provider in case database is exposed
- Makes it more difficult for attackers to fool authentication procedure

**3** Post-quantum security

**6** Transparent, Free Software solution

**9** E-money issuer does not have to protect consumer data against its own staff and can respect consumer privacy



# SOCIAL IMPACT OF ANASTASIS



Low-cost solution  
with minimal  
environmental  
impact



Increases  
informational self-  
determination by  
keeping consumers in  
control of their data



Free Software  
contributes to the  
global Commons

# OPERATING MODEL



## REVENUE

- E-money issuers pay Anastasis UG to offer service to consumers with wallets to satisfy their regulatory requirements (service must exist)
- Wallet operators pay Anastasis UG to assist with technical integration
- Consumers pay Anastasis UG for safekeeping and/or recovery (subscription)



## EXPENSES

- Development and operations (staff costs)
- Server infrastructure

# THE MARKET

(\*): This is the only entry not yet validated by letters of interest or hard commitments.



Electronic wallets for blockchain wallets and/or fiat currencies



Key store for communication keys, such as OpenPGP or X.509



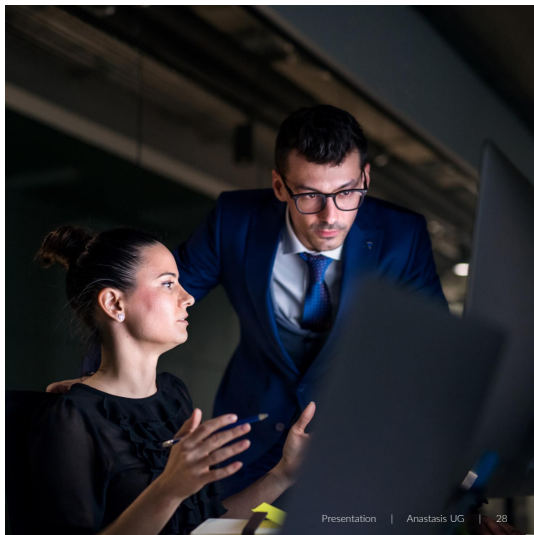
Identity management solutions



Password managers and disk encryption key material (\*)

# MAIN RISKS AND MITIGATIONS

- 1 IMPLEMENTATION RISK**  
Straightforward design simplifies work
- 2 INFORMATION SECURITY RISK**  
Privacy-by-design minimizes loss
- 3 DISTRIBUTION ON CUSTOMER SIDE**  
Strong partners with implementation need
- 4 CASH FLOW**  
Cloud-based deployment with outsourcing of procedures that amortize only at scale
- 5 USABILITY**  
Will work with UX expert



## Next Steps

- ▶ Implementation still needs:
  - ▶ ~~Escrowed wallet backup and synchronization solution~~  
⇒ Anastasis!
  - ▶ ~~Finish integration with existing banking system (EBICS, FinTS)~~ ⇒ LibFinTS
  - ▶ ~~Code security audit~~ ⇒ Exchange report now public!
  - ▶ ~~Improved design and usability~~  
⇒ Discussions with UX experts on public list!
  - ▶ Internationalization ⇒ <https://weblate.taler.net/>
  - ▶ Porting to more platforms (Web shops, Android, POS, iOS)
- ▶ Regulatory approval (withdraw and deposit limits, ~~independent auditor~~, KYC/AML process validation)
- ▶ In discussions with various (commercial and central) banks

# Visions

- ▶ Be paid to read advertising, starting with spam
- ▶ Give welfare without intermediaries taking huge cuts
- ▶ Forster regional trade via regional currencies
- ▶ Eliminate corruption by making all income visible
- ▶ Stop the mining by making crypto-currencies useless for anything but crime