Learning Objectives

How do standard RSA signatures work?

What are Blind Signatures?

How do RSA blind signatures work?

What are the main applications for Blind Signatures?
Reminder: RSA

Generate random $p, q$ primes and $e$ such that

$$GCD((p - 1)(q - 1), e) = 1$$ \hspace{1cm} (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: $m \equiv s^e \mod n$?

These equations are heavily simplified and should not be used like this in production!
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:
  \[
  D_{A_{\text{priv}}} (D_{B_{\text{priv}}} (E_{A_{\text{pub}}} (E_{B_{\text{pub}}} (m)))) \neq m
  \]  
  (2)
- PKCS#1 / RFC 3447 define a padding standard
Blind signatures with RSA [1]

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

1. Obtain public key \((e, n)\)
2. Compute 
   \[ f := FDH_n(m), \quad f < n. \]
3. Generate random 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 
   \[ s := s'b^{-1} \mod n \]
Applications for Blind Signatures

- Untraceable payments
- Unlinkable access tokens (PrivacyPass)
Provider setup: Create a denomination key (RSA)

1. Generates random primes $p, q$.
2. Computes $n := pq$, 
   \[ \phi(n) = (p - 1)(q - 1) \]
3. Picks small $e < \phi(n)$ such that 
   \[ d := e^{-1} \mod \phi(n) \text{ exists.} \]
4. Publishes public key $(e, n)$. 
Merchant setup: Create a signing key (EdDSA)

- Generates random $m \mod o$ as private key
- Computes public key $M := mG$

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

- Generates random $c \mod o$ as private key
- Computes public key $C := cG$

**Capability:** $c \Rightarrow$
1. Obtains public key \((e, n)\)
2. Computes \(f := FDH_n(C), f < n\).
3. Generates random blinding factor \(b \in \mathbb{Z}_n\)
4. Transmits \(f' := fb^e \mod n\)
1. Receives $f'$.
2. Computes $s' := f'^d \mod n$.
3. Sends signature $s'$.
Customer: Unblind signature (RSA)

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

Taler (Withdraw coins)

Customer Browser

Bank Site

Taler Exchange

HTTP

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
7 request transaction authorization
8 transaction authorization
9 withdrawal confirmation
10 execute wire transfer
11 withdraw request
12 signed blinded coins
13 unblind coins

Christian Grothoff

NEXT, GENERATION, INTERNET
Customer: Build shopping cart
Merchant: Propose contract (EdDSA)

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

transmit

Customer
Customer: Spend coin (EdDSA)

1. Receive proposal $D$, $EdDSAm(D)$.
2. Send $s$, $C$, $EdDSAc(D)$
Merchant and Provider: Verify coin (RSA)

\[ s^e \mod n \equiv FDH_n(C) \]

The provider (Taler: exchange) does not only verify the signature, but also checks that the coin was not double-spent.

GNU Taler is an online payment system.
Payment processing with Blind Signatures

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. Confirm contract
6. Forward payment
7. Send hash of digital contract and payment information
8. Forward payment
9. Navigate to fulfillment URL
10. Confirm payment
11. Reload fulfillment URL for delivery
12. Provide product resource
References I

David Chaum.

Florian Dold.
PhD thesis, University of Rennes 1, France, 2019."
Acknowledgements

Funded by the European Union (Project 101135475).

Funded by SERI (HEU-Projekt 101135475-TALER).

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union.

Neither the European Union nor the granting authority can be held responsible for them.