1. In this lecture, we will look at Blind Signatures, a cryptographic mechanism to achieve unlinkability.

2. Unlinkability is important for anonymity, as it allows us to avoid intersection attacks from an attacker linking multiple events performed by the same user.

3. Blind signatures can thus be critical to achieve anonymity.
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?

1. The goal today is to understand blind signatures.
2. We will also use this as an opportunity to review the RSA cryptosystem.
3. While RSA is uncommon in modern cryptography and deprecated for encryption, it remains a good choice for blind signatures.
Generate random $p, q$ primes and $e$ such that

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

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

Next, let's define how standard RSA signatures work:

1. $\langle e, n \rangle$ are the public key.
2. $\langle d, p, q \rangle$ are the private key.
3. $s$ is the signature over message $m$.
4. RSA-2048 seems secure for the foreseeable future.
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 $m = \sqrt[2]{C}$
  \[ \Rightarrow \text{Small } e \text{ can be bad!} \]

1. In extreme cases, $e = 3$ is mathematically possible, and some implementations use 3, 17 or 65537 as small values of $e$ improve performance!
2. How small could $m$ be? Well, 0 and 1 are perfectly valid plaintexts.
3. Note that $m^e < n$ is deliberately written without the $\mod n$. The point is that here $m^e = m^e \mod n$.
4. This breaks the cryptosystem, as $\sqrt[2]{C}$ is cheap to compute.
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

How do standard RSA signatures work?

1. Different types of padding have been a historical proposal to address low exponent issues as well as chosen plaintext attacks for RSA encryption.
2. It also helps in cases where RSA symmetry is not desired.
3. When signing, an alternative is the use of a full-domain hash $FDH_r(m)$.
4. $FDH_r(m)$ hashes $m$ cryptographically such that the result is equally distributed in the range of $[0, n)$.
What are Blind Signatures?

1. The video does not quite capture the cryptographic notion of blind signatures.
2. Just not learning the message we are signing over is insufficient to achieve unlinkability: in the example, the signer could still later recognize their signature.
3. What we actually need is an additional step by which the blind signature is transformed into a signature over the message that is unlinkable to the original blind signing process.
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. Receive \(f'\).
2. Compute \(s' := f'^d \mod n\).
3. Send \(s'\).

1. Receive \(s'\).
2. Compute \(s := s'b^{-1} \mod n\). Because multiplying with “random” values is a bijection, \(s\) is unlinkable to \(s'\) and \(f'\).
3. The unlinkability of RSA blind signatures cannot be broken computationally and is thus also quantum-safe!
4. However, the resulting signature is not quantum-safe against forgery.
Blind signatures with RSA [1]

1. Obtain public key \((e, n)\)
2. Compute \(f := FDH_n(m)\), \(f < n\).
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 \(s'\).

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

How do RSA blind signatures work?

1. Key generation for blind signatures is identical to RSA.
2. The client first computes the \(FDH_n(m)\) and blinds the hash by multiplying with \(b^e\).
3. Given that \(f\) could be any value \(\mod n\) with equal probability and the same applies to \(b\) and thus \(b^e\), \(f'\) is also equally distributed in the range \([0, n)\).
4. Signing is exactly like in vanilla RSA, without padding!
5. \(b^{e}b^{-1} = 1\), thus \(s'\) becomes \(m^2 \mod n\). Because multiplying with "random" values is a bijection, \(s\) is unlinkable to \(s'\) and \(f'\).
6. The unlinkability of RSA blind signatures cannot be broken computationally and is thus also quantum-safe!
7. However, the resulting signature is not quantum-safe against forgery.
Blind signatures with RSA [1]

1. Key generation for blind signatures is identical to RSA.
2. The client first computes the $FDH_n(m)$ and blinds the hash by multiplying with $b^e$.
3. Given that $f$ could be any value mod $n$ with equal probability and the same applies to $b$ and thus $b^e$, $f'$ is also equally distributed in the range $[0, n)$.
4. Signing is exactly like in vanilla RSA, without padding!
5. $b^{ed}b^{-1} = 1$, thus $s$ becomes $m^d$ mod $n$. Because multiplying with "random" values is a bijection, $s$ is unlinkable to $s'$ and $f'$.
6. The unlinkability of RSA blind signatures cannot be broken computationally and is thus also quantum-safe!
7. However, the resulting signature is not quantum-safe against forgery.
What are the main applications for Blind Signatures?

1. Today, we will focus on the main classical application of blind signatures, untraceable payments.
2. The basic idea is actually over 40 years old; however, we will already present it with some key improvements from [2] that we will need later when we talk about GNU Taler.
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 $d := e^{-1} \mod \phi(n)$ exists.
4. Publishes public key $(e, n)$.

---

What are the main applications for Blind Signatures?

- Provider setup: Create a denomination key (RSA)

1. We will present the mathematics on the left, and illustrating pictures on the right.
2. For now, remember that the hammer represents the private key to make blind signatures, and the seal the public key.
Merchant setup: Create a signing key (EdDSA)

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

**Capability:** $m \Rightarrow M$

What are the main applications for Blind Signatures?

1. We will also need traditional non-blind signatures for merchants and customers.
2. Here, we will just use $m$ for the private key, $M$ for the public key, and the wax seal to represent an EdDSA signature.
3. In principle, DSA or ECDSA could also be used, but EdDSA is compact and more secure.
4. Knowledge of $m$ gives the capability to create $M$-signatures.
Customer: Create a planchet (EdDSA)

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

** Capability:** $c \Rightarrow XNAGYE6P65735P4H1NGN8DT528W$

Customer: Create a planchet (EdDSA)

1. For the customer's digital cash, we also need a public-private key pair per coin.
2. We will use $c$ for the private key and $C$ for the public key.
3. Visually, we will place the (base32-encoded) public key as a (serial) number on the coin to make each coin unique.
4. Knowledge of $c$ gives the capability to create coin signatures that prove ownership over the coin.
5. Note that the image shown here is not a coin, but a planchet ("proto-coin"). After all, this is just a public key, and we need to get a signature over it to make it valueable.
Customer: Blind planchet (RSA)

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

---

What are the main applications for Blind Signatures?

1. To turn a planchet into a coin, the customer blinds it using a random blinding factor \(b\), which we will illustrate as putting the coin into an envelope that is "locked" with \(b\).
2. The result is then sent to the payment service provider, in GNU Taler called an exchange, together with instructions which account (in GNU Taler: reserve) to debit in exchange for issuing the digital cash.
1. Receives $f'$.
2. Computes $s' := f'^d \bmod n$.
3. Sends signature $s'$.

1. The exchange receives the blinded coin's public key together with information about the reserve to debit.
2. If the customer has enough funds in the reserve, a the blinded message is signed and the reserve is debited.
3. The resulting blind signature is then returned to the customer.
Customer: Unblind signature (RSA)

1. Receives $s'$.
2. Computes $s := s' b^{-1} \mod n$

---

What are the main applications for Blind Signatures?

- The customer unblinds the signature using $b$.
- The resulting signature is stored (together with $c$) in the customer's digital wallet.
Withdrawing coins on the Web

1. This figure illustrates the entire process when using it to withdraw digital cash from an online banking site.
2. The user first logs into their online banking site and asks the bank to issue digital cash.
3. The bank interacts with the Taler wallet, primarily to determine the wire transfer subject to use which will allow the wallet to claim ownership over the funds transferred to the exchange. Here, Taler uses an ephemeral reserve public-private key-pair. The reserve public key becomes the wire transfer subject.
4. After the wire transfer has been effective, the wallet can withdraw the digital coins by signing the request using the reserve private key.
What are the main applications for Blind Signatures?

- Customer: Build shopping cart

1. We will now discuss the payment process.
2. First, the customer has to choose what they want to buy.
3. In GNU Taler, this is called the order.
1. Complete proposal $D$.
2. Send $D, \text{EdDSA}_m(D)$

Customer

Merchant: Propose contract (EdDSA)

1. The merchant completes the order with additional information, such as adding the jurisdiction, applicable taxes, and how long the commercial offer is valid.
2. The merchant then signs all of this to create a commercial proposal, which is sent to the customer.

What are the main applications for Blind Signatures?
Customer: Spend coin (EdDSA)

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

Merchant

transmit

transmit

1. The customer reviews the proposal and signs it also. However, as the customer is anonymous and thus obviously not identified, they sign with their coins instead of with their identity.
2. This signature says that the owner of the coins accepts the proposal and pays for it with the value of the coins.
3. The illustration only shows one coin, but depending on the amount multiple coins would be used to sign the proposal.
4. Once both parties have signed the proposal with valid signatures, it becomes a legally binding contract: the value of the coins is owed to the merchant, and the customer is owed whatever the contract stipulates.
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.**

---

What are the main applications for Blind Signatures?

1. Finally, both the merchant and the provider check all the signatures, especially the one showing that the coin is valid. This is the first time anyone but the customer sees the coin’s public key, and it is the last transaction the coin should be used for.
2. Naturally, a customer could try to spend a coin more than once, so the provider MUST keep records of all spent coins to prevent double-spending.
3. The coin’s signature under a previous contract serves as proof of an attempt to double-spend.
4. The merchant must check with the provider against double-spending before accepting the contract and fulfilling their part of the contract.
**Payment processing with Blind Signatures**

1. This figure illustrates the payment process for digital goods with GNU Taler on the Web.
2. First the customer goes to an **offer URL** where the Taler payment method is chosen and subsequently the wallet is triggered.
3. The wallet **claims** the proposal and asks the user to confirm the payment. It sends the payment data to the merchant.
4. Finally, the customer is directed to the **fulfillment URL** where they are provided with either the digital goods they purchased, or other ways to access information about the business logic they triggered, such as shipment tracking.
References

David Chaum. 

Florian Dold. 
The GNU Taler system: practical and provably secure electronic payments. (Le système GNU Taler: Paiements électroniques pratiques et sécurisés). 
Acknowledgements

Funded by the European Union (Project 101135475).

Co-funded by the European Union

Project funded by

- Schweizerische Eidgenossenschaft
- Confederation swiss
- Confederazione Svizzera
- Confederate svizzera
- Swiss Confederation
- Federal Department of Economic Affairs, Education and Research SECO
- State Secretariat for Education, Research and Innovation SERI

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.