IRMA’s Idemix core

Understanding the crypto behind selective, unlinkable attribute disclosure

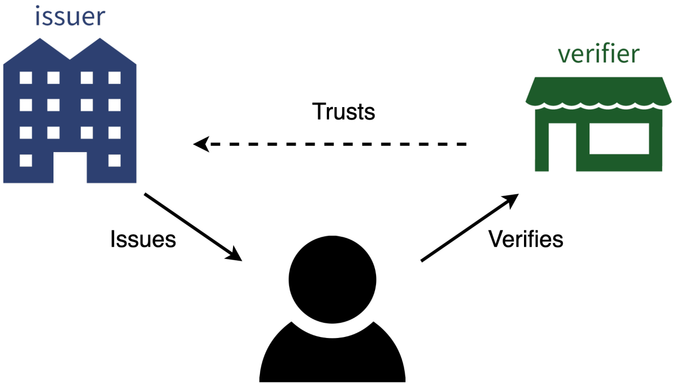
Maja Reißner
Sietse Ringers

July 24, 2022

Sietse Ringers

Maja Reißner

Self-Sovereign Identity



The offline world, using a paper diploma:

Diploma

Sietse Ringers,
born on **July, 11th 1984**,
has earned a **PhD** at
University of Groningen.

sign

The offline world, using a paper diploma:

Diploma

xxxxxxxxxxx,
born on **xxxxxxxxxxx**,
has earned a **PhD** at
xxxxxxxxxxx.

sign

- *Feature:* Selective disclosure

The offline world, using a paper diploma:

Diploma

xxxxxxxxxxx,
born on **xxxxxxxxxxx**,
has earned a **PhD** at
xxxxxxxxxxx.

sign

- *Feature:* Selective disclosure
- *Problem:* Replay attacks

The offline world, using a paper diploma:

Diploma

zzzzzzzzzzz,
born on **zzzzzzzzzzz**,
has earned a **PhD** at
zzzzzzzzzzz.

sign

- *Feature:* Selective disclosure
- *Problem:* Replay attacks
- *Feature:* Unlinkability

Contents of this talk

Goal is to understand:

```go
func (s *CLSignature) Verify(pk *gabikeys.PublicKey, ms []*big.Int) bool {
	// Q = A^e * R * S^v
	Ae := new(big.Int).Exp(s.A, s.E, pk.N)
	R, err := RepresentToPublicKey(pk, ms)
	if err != nil {
		return false
	}
	if s.KeyshareP != nil {
		R.Mul(R, s.KeyshareP)
	}
	Sv, err := common.ModPow(pk.S, s.V, pk.N)
	if err != nil {
		return false
	}
	Q := new(big.Int).Mul(Ae, R)
	Q.Mul(Q, Sv).Mod(Q, pk.N)

	// Signature verifies if Q == Z
	return pk.Z.Cmp(Q) == 0
}
```

Contents of this talk

Goal is to understand:

```
func (s *CLSignature) Verify(pk *gabikeys.PublicKey, ms []*big.Int) bool {
	// Q = A^e * R * S^v
	Ae := new(big.Int).Exp(s.A, s.E, pk.N)
	R, err := RepresentToPublicKey(pk, ms)
	if err != nil {
		return false
	}
	if s.KeyshareP != nil {
		R.Mul(R, s.KeyshareP)
	}
	Sv, err := common.ModPow(pk.S, s.V, pk.N)
	if err != nil {
		return false
	}
	Q := new(big.Int).Mul(Ae, R)
	Q.Mul(Q, Sv).Mod(Q, pk.N)

	// Signature verifies if Q == Z
	return pk.Z.Cmp(Q) == 0
}
```

$$\tilde{A}^e = \frac{Z}{S^{\tilde{v}} \prod_{i=0}^{k} R_i^{a_i}} \mod n$$

Contents of this talk

Goal is to understand:

```
func (s *CLSignature) Verify(pk *gabikeys.PublicKey, ms []*big.Int) bool {
	// Q = A^e * R * S^v
	Ae := new(big.Int).Exp(s.A, s.E, pk.N)
	R, err := RepresentToPublicKey(pk, ms)
	if err != nil {
		return false
	}
	if s.KeyshareP != nil {
		R.Mul(R, s.KeyshareP)
	}
	Sv, err := common.ModPow(pk.S, s.V, pk.N)
	if err != nil {
		return false
	}
	Q := new(big.Int).Mul(Ae, R)
	Q.Mul(Q, Sv).Mod(Q, pk.N)

	// Signature verifies if Q == Z
	return pk.Z.Cmp(Q) == 0
}
```

$$\tilde{A}^e = \frac{Z}{S^{\tilde{v}} \prod_{i=0}^{k} R_i^{a_i}} \mod n$$

- Selective disclosure
  - Distinguish attributes
  - Hide attributes
- Ownership of the credential
- Unlinkability
- Disclosure of multiple credentials

Distinguishing attributes

Textbook RSA:

$$A = M^d \bmod n$$

Verify with:

$$A^e = M \bmod n$$

Diploma

Sietse Ringers
has earned a PhD.

sign

Distinguishing attributes

Textbook RSA:

$$A = M^d \bmod n$$

Verify with:

$$A^e = M \bmod n$$

Signature scheme which differentiates attributes:

$$M = \frac{Z}{S^v R_1^{a_1} R_2^{a_2}}$$

Diploma

Sietse Ringers
has earned a PhD.

sign

Diploma

name: **Sietse Ringers**,
title: **PhD**

A, e, v

Distinguishing attributes

Textbook RSA:

$$A = M^d \bmod n$$

Verify with:

$$A^e = M \bmod n$$

Signature scheme which differentiates attributes:

$$M = \frac{Z}{S^v R_1^{a_1} R_2^{a_2}}$$

$$\boldsymbol{A^e = \frac{Z}{S^v R_1^{a_1} R_2^{a_2}} \bmod n}$$

Diploma

Sietse Ringers
has earned a PhD.

sign

Diploma

name: **Sietse Ringers**,
title: **PhD**

A, e, v

Distinguishing attributes

Textbook RSA:

$$A = M^d \bmod n$$

Verify with:

$$A^e = M \bmod n$$

Signature scheme which differentiates attributes:

$$M = \frac{Z}{S^v R_1^{a_1} R_2^{a_2}}$$

$$\boldsymbol{A^e = \frac{Z}{S^v R_1^{a_1} R_2^{a_2}} \bmod n}$$

→ **Camenisch-Lysyanskaya signature scheme**

Diploma

Sietse Ringers
has earned a PhD.

sign

Diploma

name: **Sietse Ringers**,
title: **PhD**

A, e, v

Issuer setup

Choose private key p, q so that $n = pq$

Choose constants $Z, S, R_1, \ldots, R_i$

Share public key $(Z, S, R_1, \ldots, R_i, n)$

During issuance of a credential

Choose e, v

Calculate

$$A = \left(\frac{Z}{S^v R_1^{a_1} R_2^{a_2}} \right)^{e^{-1}} \bmod n$$

Share signature (A, e, v)

Verification equation

$$\boldsymbol{A^e = \frac{Z}{S^v R_1^{a_1} R_2^{a_2}} \bmod n}$$

Signature verification:

$$A^e = \frac{Z}{S^v R_1^{a_1} R_2^{a_2}}$$

$$H = R_1^{a_1}$$

Diploma

name: **xxxxxxxx**,

title: **PhD**

A, e, v

Signature verification:

$$A^e = \frac{Z}{S^v H R_2^{a_2}}$$

$$H = R_1^{a_1}$$

Diploma

name: **xxxxxxxx**,

title: **PhD**

A, e, v

Problem: Forgery of other attributes.

New example, given:

a_1 ... name (to be hidden)

a_2 ... age $= 17$

Then I could forge my age to 18, like this:

claim $a_2 = 18$; $\quad H' = HR_2^{-1}$

$$A^e = \frac{Z}{S^v H' R_2^{a_2}} = \frac{Z}{S^v (H \cdot R_2^{-1}) R_2^{a_2}} = \frac{Z}{S^v H R_2^{a_2 - 1}}$$

Hiding an attribute

Signature verification:

$$A^e = \frac{Z}{S^v H R_2^{a_2}}$$

$$H = R_1^{a_1}$$

Diploma

name: **xxxxxxxx**,

title: **PhD**

A, e, v

Schnorr's Zero Knowledge (ZK) protocol, given $H = R^a$

choose random t			
$U = R^t \bmod n$	$\xrightarrow{U}$		**commitment**
	$\xleftarrow{c}$	choose random c	**challenge**
$r = t + ca$	$\xrightarrow{r}$	$R^r H^{-c} \stackrel{?}{=} U \bmod n$	**response**

Proof why Schnorr's Zero Knowledge protocol works

$$\begin{array}{rcll}
\text{given } H = R^a & & & \\
\text{choose random } t & & & \\
U = R^t \bmod n & \xrightarrow{U} & & \textbf{commitment} \\
 & \xleftarrow{c} & \text{choose random } c & \textbf{challenge} \\
r = t + ca & \xrightarrow{r} & R^r H^{-c} \stackrel{?}{=} U \bmod n & \textbf{response}
\end{array}$$

$$\begin{aligned}
R^r \cdot H^{-c} & \\
&= R^{t+ca} \cdot H^{-c} \\
&= R^t \cdot R^{ca} \cdot H^{-c} \\
&= R^t \cdot R^{ca} \cdot (R^a)^{-c} \\
&= R^t \cdot R^{ca} \cdot R^{-ca} \\
&= R^t \cdot \cancel{R^{ca}} \cdot \cancel{R^{-ca}} \\
&= R^t \qquad\qquad\qquad = U
\end{aligned}$$

Ownership of the credential

given $H = R^a$			
choose random t			
$U = R^t \bmod n$	$\xrightarrow{U}$		**commitment**
	$\xleftarrow{c}$	choose random c	**challenge**
$r = t + ca$	$\xrightarrow{r}$	$R^r H^{-c} \stackrel{?}{=} U \bmod n$	**response**

Ownership of the credential

given $H = R^a$			
choose random t			
$U = R^t \bmod n$	$\xrightarrow{U}$		**commitment**
	$\xleftarrow{c}$	choose random c	**challenge**
$r = t + ca$	$\xrightarrow{r}$	$R^r H^{-c} \stackrel{?}{=} U \bmod n$	**response**

Diploma

name: **xxxxxxxx**,

title: **PhD**

A, e, v

Ownership of the credential

given $H = R^a$			
choose random t			
$U = R^t \bmod n$	$\xrightarrow{U}$		**commitment**
	$\xleftarrow{c}$	choose random c	**challenge**
$r = t + ca$	$\xrightarrow{r}$	$R^r H^{-c} \stackrel{?}{=} U \bmod n$	**response**

Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

A, e, v

Unlinkability

$$A^e = \frac{Z}{S^v R_0^{\text{[redacted]}} R_1^{\text{[redacted]}} R_2^{a_2}}$$

- Proof of knowledge of a_0 and a_1

Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

A, e, v

Unlinkability

$$A^{\blacksquare} = \frac{Z}{S^{\blacksquare} R_0^{\blacksquare} R_1^{\blacksquare} R_2^{a_2}}$$

- Proof of knowledge of a_0 and a_1
- Also hide and proof of knowledge of e and v

Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

A, x, x

Unlinkability

$$A^{\blacksquare} = \frac{Z}{S^{\blacksquare} R_0^{\blacksquare} R_1^{\blacksquare} R_2^{a_2}}$$

- Proof of knowledge of a_0 and a_1
- Also hide and proof of knowledge of e and v

Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

A, x, x

Making A unlinkable:

Unlinkability

$$A^{\blacksquare} = \frac{Z}{S^{\blacksquare} R_0^{\blacksquare} R_1^{\blacksquare} R_2^{a_2}}$$

- Proof of knowledge of a_0 and a_1
- Also hide and proof of knowledge of e and v

Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

A, x, x

Making A unlinkable:

- Choose random number r

Unlinkability

$$A^{\blacksquare} = \frac{Z}{S^{\blacksquare} R_0^{\blacksquare} R_1^{\blacksquare} R_2^{a_2}}$$

- Proof of knowledge of a_0 and a_1
- Also hide and proof of knowledge of e and v

Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

$\tilde{A}, x, x$

Making A unlinkable:

- Choose random number r
- Set

$$\tilde{A} = AS^r \bmod n \qquad \tilde{v} = v - er$$

$$A^{\blacksquare} = \frac{Z}{S^{\blacksquare} R_0^{\blacksquare} R_1^{\blacksquare} R_2^{a_2}}$$

- Proof of knowledge of a_0 and a_1
- Also hide and proof of knowledge of e and v

Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

$\tilde{A}, x, x$

Making A unlinkable:

- Choose random number r
- Set

$$\tilde{A} = AS^r \bmod n \qquad \tilde{v} = v - er$$

- Then:

$$\tilde{A}^e = (AS^r)^e = A^e S^{er} = \frac{Z}{S^v R_1^{a_1} R_2^{a_2}} S^{er} = \frac{Z}{S^{v-er} R_1^{a_1} R_2^{a_2}} = \frac{Z}{S^{\tilde{v}} R_1^{a_1} R_2^{a_2}}$$

Overview: RSA $\Rightarrow$ Camenisch-Lysyanskaya

$$A^e = M \bmod n$$

$$\Downarrow$$

$$A^e = \frac{Z}{S^v R_0^{a_0} R_1^{a_1} R_2^{a_2}} \bmod n$$

Diploma

Sietse Ringers
has earned a PhD.

sign

Diploma

secret: **xxxxxxxx**,
name: **Sietse Ringers**,
title: **PhD**

A, e, v

The full disclosure protocol

Attributes: (a_0, a_1, a_2) Signature: (A, e, v)

- Set $\tilde{A} = AS^r \bmod n$ $\tilde{v} = v - er$

Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

$\tilde{A}, x, x$

The full disclosure protocol

Attributes: (a_0, a_1, a_2) Signature: (A, e, v)

- Set $\tilde{A} = AS^r \bmod n$ $\tilde{v} = v - er$

$$\tilde{A}^e = \frac{Z}{S^{\tilde{v}} R_0^{a_0} R_1^{a_1} R_2^{a_2}}$$

Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

$\tilde{A}, x, x$

The full disclosure protocol

Attributes: (a_0, a_1, a_2) Signature: (A, e, v)

- Set $\tilde{A} = AS^r \bmod n$ $\tilde{v} = v - er$

$$\tilde{A}^e = \frac{Z}{S^{\tilde{v}} R_0^{a_0} R_1^{a_1} R_2^{a_2}}$$

$$ZR_2^{-a_2} = \tilde{A}^e S^{\tilde{v}} R_0^{a_0} R_1^{a_1}$$

Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

$\tilde{A}, x, x$

The full disclosure protocol

Attributes: (a_0, a_1, a_2) Signature: (A, e, v)

- Set $\tilde{A} = AS^r \bmod n$ $\tilde{v} = v - er$

$$\tilde{A}^e = \frac{Z}{S^{\tilde{v}} R_0^{a_0} R_1^{a_1} R_2^{a_2}}$$

$$ZR_2^{-a_2} = H = \tilde{A}^e S^{\tilde{v}} R_0^{a_0} R_1^{a_1}$$

Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

$\tilde{A}, x, x$

The full disclosure protocol

Attributes: (a_0, a_1, a_2) Signature: (A, e, v)

- Set $\tilde{A} = AS^r \bmod n$ $\tilde{v} = v - er$

$$\tilde{A}^e = \frac{Z}{S^{\tilde{v}} R_0^{a_0} R_1^{a_1} R_2^{a_2}}$$

$$ZR_2^{-a_2} = H = \tilde{A}^e S^{\tilde{v}} R_0^{a_0} R_1^{a_1}$$

- Choose random $t_e, t_{\tilde{v}}, t_0, t_1$

Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

$\tilde{A}, x, x$

The full disclosure protocol

Attributes: (a_0, a_1, a_2) Signature: (A, e, v)

- Set $\tilde{A} = AS^r \bmod n$ $\tilde{v} = v - er$

$$\tilde{A}^e = \frac{Z}{S^{\tilde{v}} R_0^{a_0} R_1^{a_1} R_2^{a_2}}$$

$$ZR_2^{-a_2} = H = \tilde{A}^e S^{\tilde{v}} R_0^{a_0} R_1^{a_1}$$

- Choose random $t_e, t_{\tilde{v}}, t_0, t_1$
- Perform the following protocol:

$$U = \tilde{A}^{t_e} S^{t_{\tilde{v}}} R_0^{t_0} R_1^{t_1} \bmod n \quad \xrightarrow{\tilde{A}, U}$$

$$\xleftarrow{c} \quad \text{choose random } c$$

$$\begin{aligned} r_e &= t_e + ce, & r_{\tilde{v}} &= t_{\tilde{v}} + c\tilde{v} \\ r_0 &= t_0 + ca_0, & r_1 &= t_1 + ca_1 \end{aligned} \quad \xrightarrow{r_i} \quad U \stackrel{?}{=} \tilde{A}^{r_e} S^{r_{\tilde{v}}} R_0^{r_0} R_1^{r_1} H^{-c} \bmod n$$

Diploma

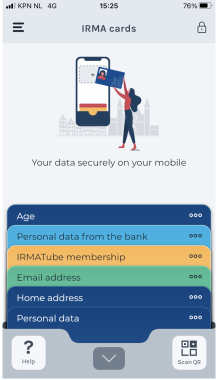
secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

$\tilde{A}, x, x$

Idemix



Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

A, e, v

What if Maja gains control over my wallet?
Can she disclose "maja@irma.app" and "PhD"?

Email address

secret: **xxxxxxxx**,
Email address: **maja@irma.app**

A, e, v

Diploma

secret: **xxxxxxxx**,
name: **xxxxxxxx**,
title: **PhD**

A, e, v

Idemix - preventing credential pooling attacks

$$U = \tilde{A}^{t_e} S^{t_{\tilde{v}}} R_0^{t_0} R_1^{t_1} \quad \xrightarrow{\tilde{A},U}$$

$$\xleftarrow{c} \quad \text{choose random } c$$

$$\begin{aligned} r_e &= t_e + ce, \quad & r_{\tilde{v}} &= t_{\tilde{v}} + c\tilde{v} \\ r_0 &= t_0 + ca_0, \quad & r_1 &= t_1 + ca_1 \end{aligned} \quad \xrightarrow{r} \quad U \stackrel{?}{=} \tilde{A}^{r_e} S^{r_{\tilde{v}}} R_0^{r_0} R_1^{r_1} H^{-c}$$

Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

$\tilde{A}, x, x$

Idemix - preventing credential pooling attacks

$$U \xrightarrow{\tilde{A}, U}$$

$$\xleftarrow{c} \quad \text{choose random } c$$

$$r_i = t_i + ca_i \xrightarrow{r_i}$$

Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

$\tilde{A}, x, x$

Idemix - preventing credential pooling attacks

$$U, U \xrightarrow{\tilde{A}, U, \tilde{A}, U}$$

$$\xleftarrow{c} \quad \text{choose random } c$$

$$r_i = t_i + c a_i, \quad r_i = t_i + c a_i \xrightarrow{r_i, r_i}$$

Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

$\tilde{A}, x, x$

Email address

secret: **xxxxxxxx**,

Email address:

sietse@irma.app

$\tilde{A}, x, x$

Idemix - preventing credential pooling attacks

Secret key:

$$r_0 = t_0 + ca_0, \quad r_0 = t_0 + ca_0$$

Diploma

secret: **xxxxxxxx**,

name: **xxxxxxxx**,

title: **PhD**

$\tilde{A}, x, x$

Email address

secret: **xxxxxxxx**,

Email address:

sietse@irma.app

$\tilde{A}, x, x$

Idemix - preventing credential pooling attacks

Secret key:

$$r_0 = t_0 + c a_0, \quad r_0 = t_0 + c a_0$$
$$\Downarrow$$
$$r_0 = t_0 + c a_0$$

User:

- Use same a_0 in each credential
- When disclosing attributes from multiple credentials, use same t_0

Verifier:

- For each credential, check that the same r_0 is used

Diploma

secret: **xxxxxxxx**,
name: **xxxxxxxx**,
title: **PhD**

$\tilde{A}, x, x$

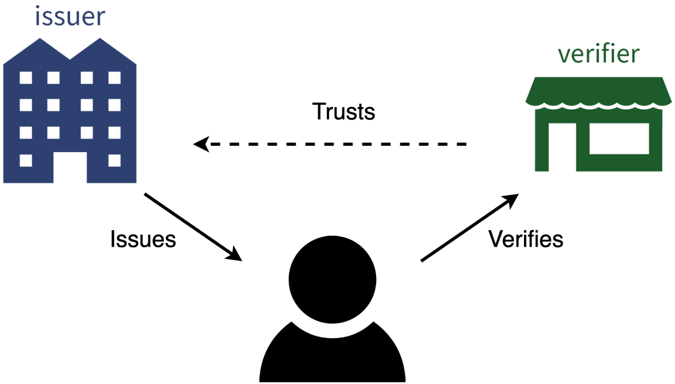
Email address

secret: **xxxxxxxx**,
Email address:
sietse@irma.app

$\tilde{A}, x, x$

Back to the beginning - the total picture

Self-Sovereign Identity



The end.

Links

Some references:

- Idemix
  - IBM Idemix spec: https://dominoweb.draco.res.ibm.com/reports/rz3730_revised.pdf
  - Idemix fact sheet: https://privacybydesign.foundation/pdf/Idemix_overview.pdf
- Camenisch-Lysyanskaya
  - Initial paper: https://www.researchgate.net/profile/Jan-Camenisch/publication/220922101_A_Signature_Scheme_with_Efficient_Protocols/links/5a8c8709458515a4068ae0a3/A-Signature-Scheme-with-Efficient-Protocols.pdf?origin=publication_detail
- Schnorr protocol
  - Initial paper: https://link.springer.com/content/pdf/10.1007/BF00196725.pdf
- IRMA
  - IRMA documentation: https://www.irma.app/docs/what-is-irma/
  - IRMA's Idemix implementation: https://github.com/privacybydesign/gabi/

More slides

in case of time or questions

valid signature 1:

$$Z = A^e S^v R_1^{a_1} \cdots R_k^{a_k}$$

valid signature 2:

$$Z = A'^e S^{\tilde{v}} R_1^{a_1'} \cdots R_k^{a_k'}$$

Combined you get

$$(A/A')^{-e} = S^{v-\tilde{v}} R_1^{a_1 - a_1'} \cdots R_k^{a_k - a_k'}$$

This is a valid signature over the attributes

$$a_1 - a_1', \ldots, a_k - a_k$$

Camenisch-Lysyanskaya's Z to prevent forgeability

Formula without Z:

$$A^e = \frac{1}{S^v R_1^{a_1} R_2^{a_2}} \bmod n$$

Example: Forgery of age attributes.

a_1 ... name (to be hidden)
a_2 ... age $= 9$

Then I could forge my age to 18, like this:
claim $a_2 = 18$; $\quad a_1' = 2a_1; A' = A^2; v' = 2v$

$$A'^e = (A^e)^2 = \left(\frac{1}{S^v R_1^{a_1} R_2^{a_2}}\right)^2 = \frac{1^2}{S^{2v} R_1^{2a_1} R_2^{2a_2}} = \frac{1}{S^{v'} R_1^{a_1'} R_2^{a_2'}} \bmod n$$

RSA prerequisite on one slide

Textbook RSA:

$$A = M^d \bmod n$$

Verify with:

$$A^e = (M^d)^e = M^{de} = M \bmod n$$

Choose primes p,q and calculate $n = pq$

Choose e, calculate inverse so that $e \cdot d = 1 \bmod \phi(n)$ so that $M^{e \cdot d} = M \bmod n$

Two mathematical hard problems:

1. Discrete logarithm problem: Just knowing A and M it's infeasible to find d.
2. Calculating the inverse of e is hard if you don't know the group order. (It's easy to calculate the group order if you know the prime factorization of n.)

Keyshare protocol

Disclosure session with split secret

$$\tilde{A}^e = \frac{Z}{S^{\tilde{v}} R_0^{s_k} R_0^{s_u} R_1^{a_1} R_2^{a_2}}$$

where the secret $a_0 = s_k s_u$

s_k is only known to the keyshare server

s_u is only known to the app user

Both s_k and s_u are ALWAYS hidden in Zero Knowledge

To perform the Schnorr protocol with the keyshare server, the app user must provide her PIN

TTP		**User**		**Issuer**
$N, R; m_t$		$N, R; m_u, P_t$	$\xleftrightarrow{\text{Issuance}}$	N, R
		Choose random w_u	$\xleftarrow{\text{nonce } \eta}$	
		$W_u = R^{w_u} \bmod N$		
Choose random w_t	$\xleftarrow{P_u, h_W}$	$h_W = \mathcal{H}(W_u)$		
$W_t = R^{w_t} \bmod N$	$\xrightarrow{W_t}$	$W = W_u W_t$		
		$c = \mathcal{H}(\eta, P, W)$		
verify $h_W = \mathcal{H}(W_u)$	$\xleftarrow{\eta, c, s_u, W_u}$	$s_u = w_u + c m_u$		
verify $R^{s_u} = P_u^c W_u \bmod N$				
$P = P_u P_t,\ W = W_u W_t$				
verify $c = \mathcal{H}(\eta, P, W)$				
$s_t = w_t + c m_t$				
$s = s_u + s_t$				
$\sigma = \text{sign}(\text{sk}, (c, s))$	$\xrightarrow{\sigma, s}$		$\xrightarrow{\sigma, c, s, P}$	verify $(\text{pk}, \sigma, (c, s))$
				$W = R^s P^{-c} \bmod N$
				verify $c = \mathcal{H}(\eta, P, W)$

[Figure credits: see notes]

Revocation of credentials - Issuance

The current accumulator is a number $\nu \in QR_n$. The first accumulator is randomly chosen by the issuer from QR_n. During issuance, the issuer

1. generates a prime e,
2. embeds the prime e as an attribute within the credential being issued,
3. uses its private key to compute $u = \nu^{1/e \bmod pq}$, and sends the tuple (u, e) to the app along with the credential,
4. stores the number e in a database for later revocation.

The revocation witness is the tuple (u, e). By definition it is valid only if $u^e = \nu \bmod n$. When using revocation, the app now proves the following to the verifier:

- "I possess a valid credential containing the disclosed attributes as well as an undisclosed attribute e."
- "I know a number u which is such that $u^e = \nu \bmod n$."

Revocation of credentials - Revocation

Compute new accumulator value:

$$\nu_{i+1} = \nu_i^{1/\tilde{e} \bmod pq}$$

Update witness:

$$u_{i+1} = u_i^b \nu_{i+1}^a$$

Proof that update mechanism works:

$$u_{i+1}^e = (u_i^b \nu_{i+1}^a)^e = u_i^{be} \nu_{i+1}^{ae} = \nu_i^b \nu_i^{ae/\tilde{e}} = (\nu_i^{b\tilde{e}} \nu_i^{ae})^{1/\tilde{e}} = (\nu_i^{b\tilde{e}+ae})^{1/\tilde{e}} = \nu_i^{1/\tilde{e}} = \nu_{i+1}$$