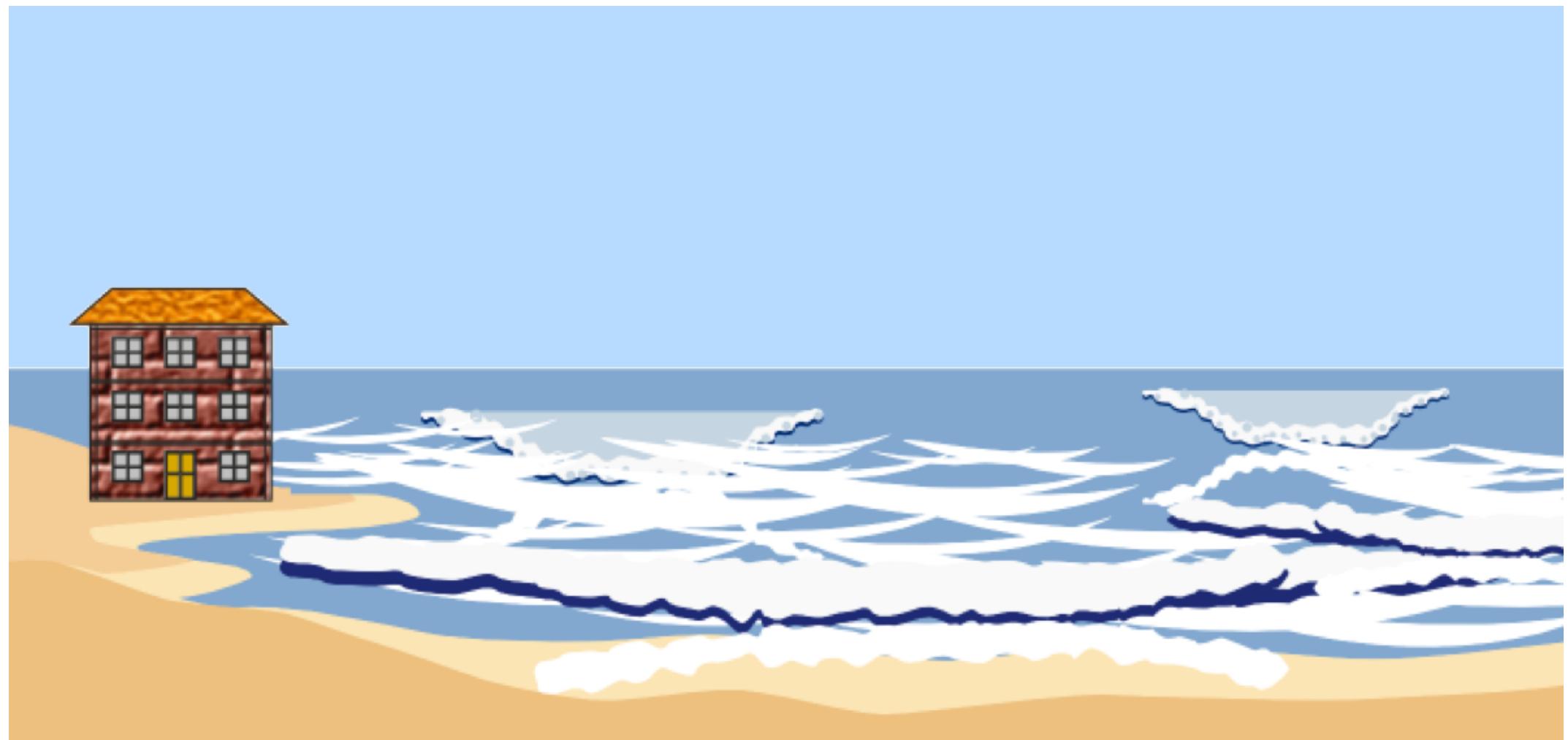




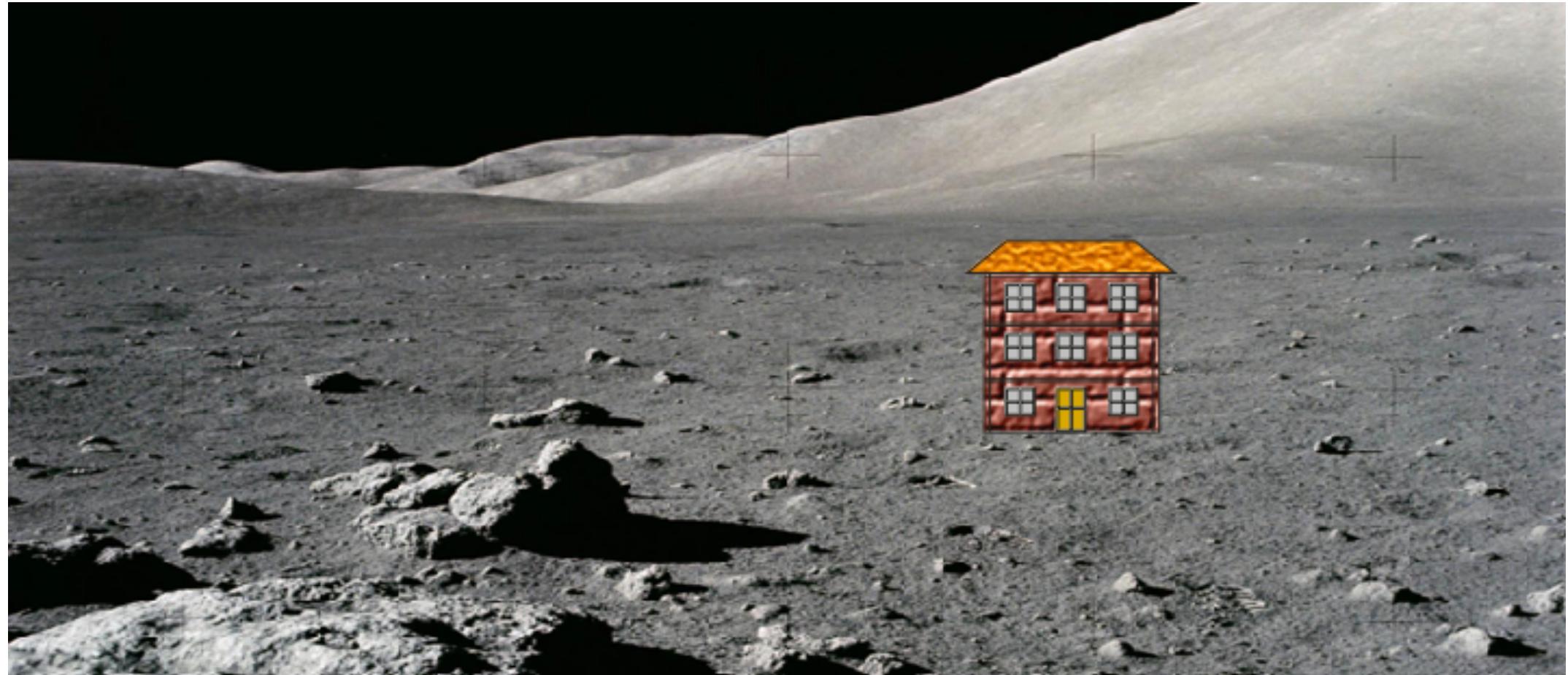
# Zero Knowledge Proofs for Constructing Protocols

Jan Camenisch  
DFINITY

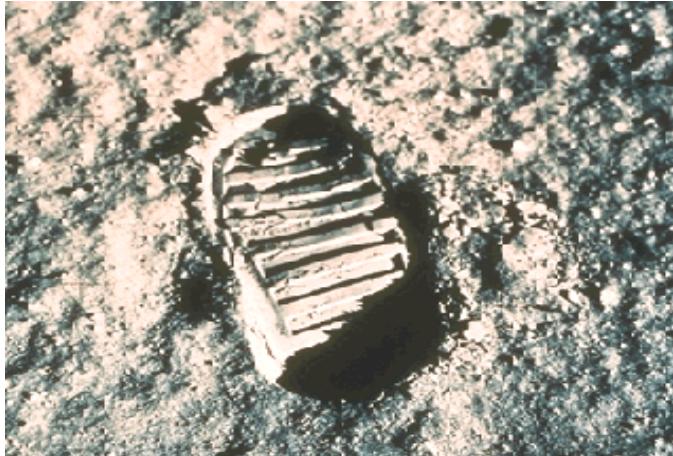
# Cryptography is a constructive science



... but end up doing this



# Computers never forget



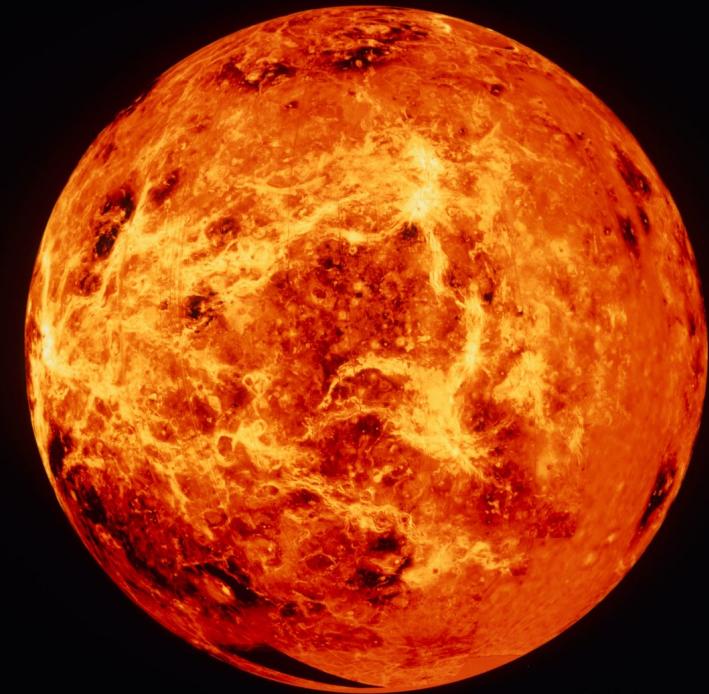
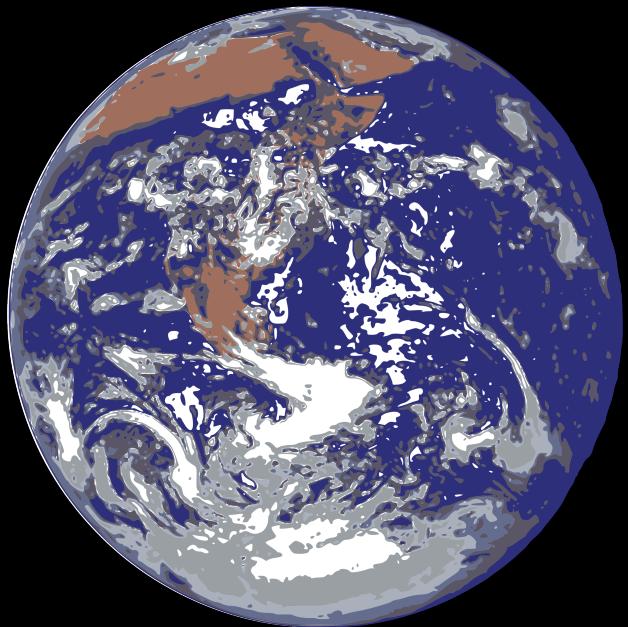
- Data is stored by default
- Data mining gets ever better
- Apps built to use & generate (too much) data
- New (ways of) businesses using personal data



- Humans forget most things too quickly
- Paper collects dust in drawers

*But that's how we design and build applications!*

# Apps securely deployable in any environments



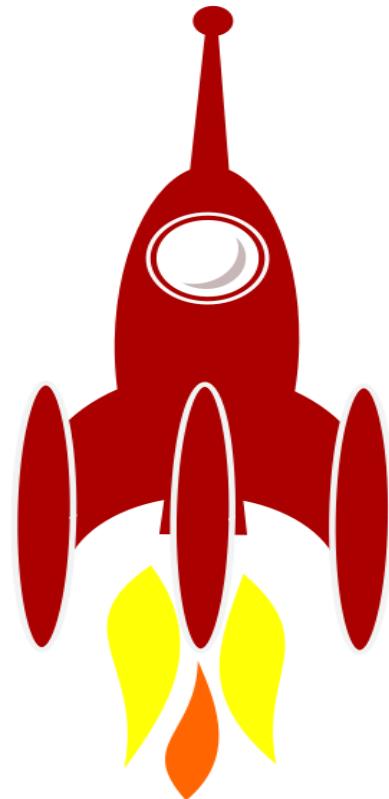
# Building & proving secure a cryptographic protocol is hard



- Finding the right abstraction
- Building an *efficient* realisation modularly from primitives
- Prove security of realisation
- Instantiate primitives to obtain protocol

Cryptography is one of the main ingredients to space faring!

Let's talk about:  
Zero knowledge proofs  
in space faring

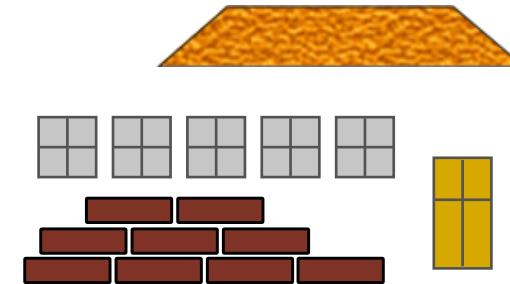


## Part I

Zero-knowledge proofs and other protocol building blocks

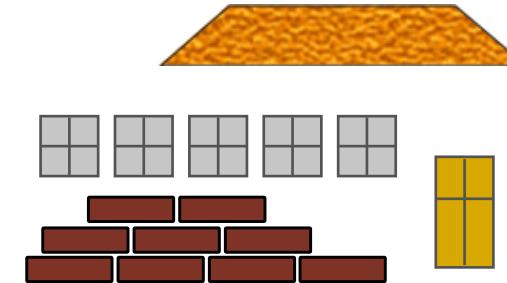
# Well defined primitives as building blocks

- Signature Schemes
- Encryption Schemes
- Commitment Schemes
- Verifiable Pseudo Random Functions
- ....



# Well defined primitives as building blocks

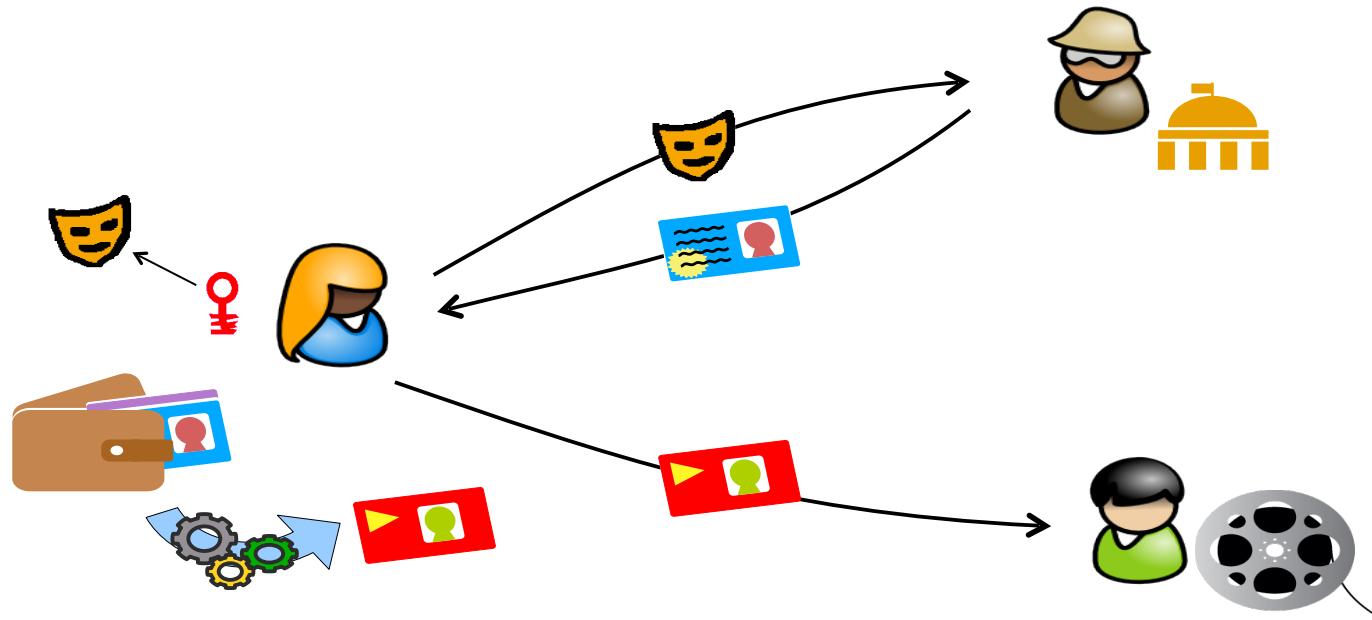
- Signature Schemes
- Encryption Schemes
- Commitment Schemes
- Verifiable Pseudo Random Functions
- ....



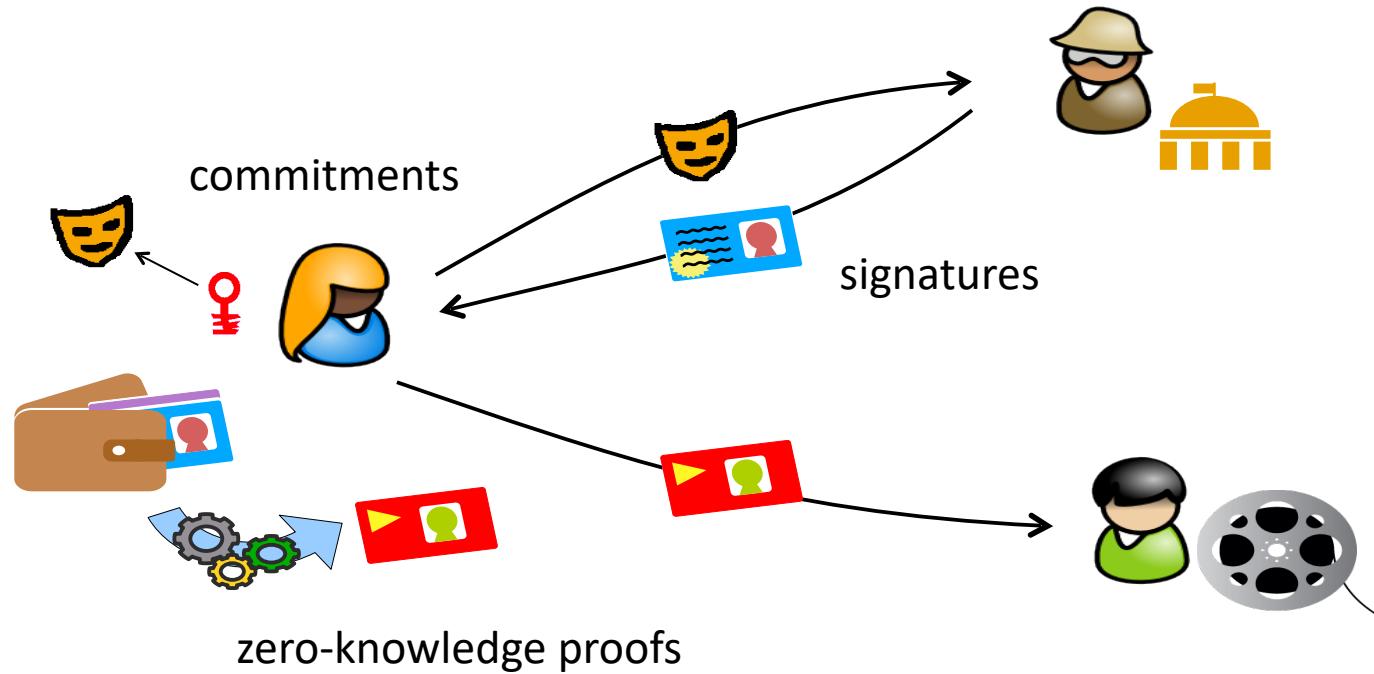
..and, of course, Zero-Knowledge Proofs as mortar!



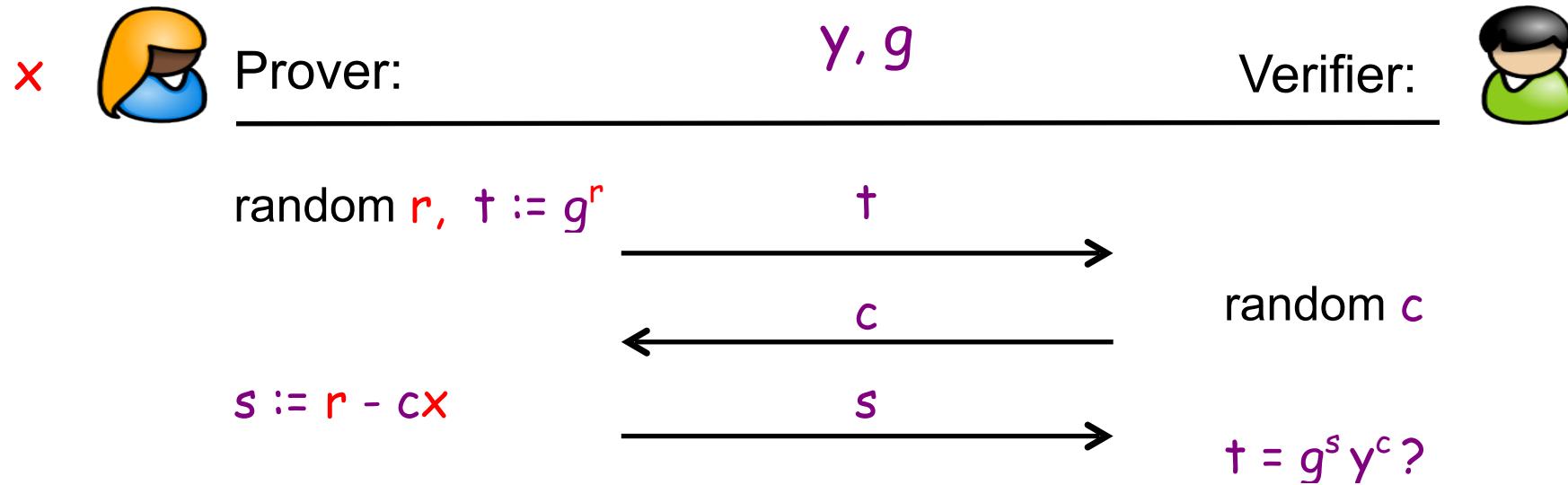
# Example application: privacy-protecting authentication



# Example application: privacy-protecting authentication



# Efficient Zero-Knowledge Proofs



Proof of knowledge: from a successful prover, *using rewinding*, one can extract  $x$  s.t.  $y = g^x$   
thus notation:  $\text{PK}\{(\text{x}): \quad y = g^x\}$

Zero knowledge: the verifier learns nothing from interacting in protocol with prover

Non-interactive with Fiat-Shamir heuristic, i.e.,  $c = H(t, y, g)$

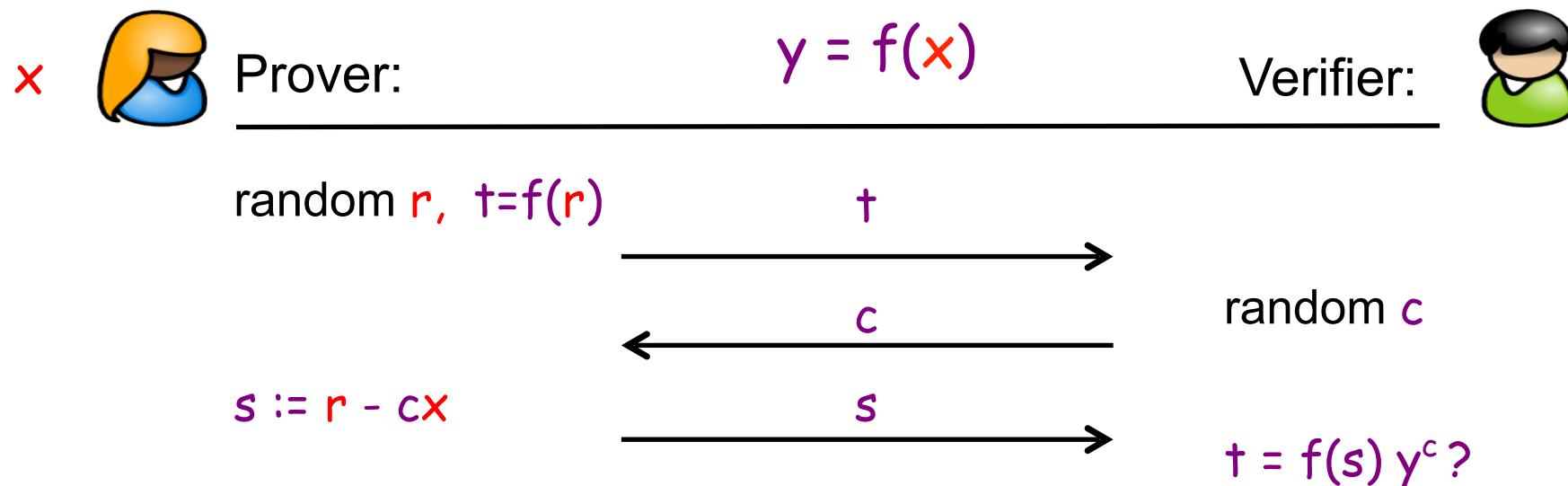


# Efficient Zero-Knowledge Proofs



Important observations:

1. Protocol works for any group homomorphism  $f: G_1 \rightarrow G$
2. Prover and verifier have work equivalent to one evaluation of  $f$



# Efficient Zero-Knowledge Proofs



Protocol works for *any group homomorphism*  $f: G_1 \rightarrow G_2$

- Classical examples

$$x \rightarrow g^x$$

$$(u, v) \rightarrow g^u h^v$$

$$(u, v) \rightarrow (k^v, g^u h^v, g^v)$$

- Bilinear maps  $e: G_1 \times G_1 \rightarrow G_t$

$$x \rightarrow e(g, g)^x$$

$$z \rightarrow e(z, g)$$

- But not

$$(u, v) \rightarrow g^{u v}$$

$$(u, v) \rightarrow e(k^v, g^u h^v)$$

or  $(x, z) \rightarrow e(z, g^x)$

Logical operators work as well:

- $\text{PK}\{(x, u, v, z): a = g^x \wedge b = e(z, g) \wedge d = g^u h^v\}$
- $\text{PK}\{(x, u, v, z): a = g^x \vee (b = e(z, g) \wedge d = g^u h^v)\}$



# Efficient Zero-Knowledge Proofs



Protocol works for *any group homomorphism*  $f: G_1 \rightarrow G_2$

- Classical examples

$$x \rightarrow g^x$$

$$(u, v) \rightarrow g^u h^v$$

$$(u, v) \rightarrow (k^v, g^u h^v, g^v)$$

- Bilinear maps  $e: G_1 \times G_1 \rightarrow G_t$

$$x \rightarrow e(g, g)^x$$

$$z \rightarrow e(z, g)$$

- But not

$$(u, v) \rightarrow g^{u v}$$

$$(u, v) \rightarrow e(k^v, g^u h^v)$$

or  $(x, z) \rightarrow e(z, g^x)$

Logical operators work as well:

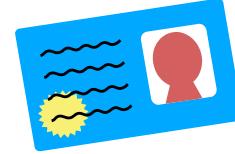
- $\text{PK}\{(x, u, v, z): a = g^x \wedge b = e(z, g) \wedge d = g^u h^v\}$

- $\text{PK}\{(x, u, v, z): a = g^x \vee (b = e(z, g) \wedge d = g^u h^v)\}$

Can be standardised! :-)



# Example signature scheme



Goal: Want to have  $\text{PK}\{(s,m)\}$ :  $1 = \text{verify\_signature}(\text{PK}, s, m)$

Take, e.g., Pointcheval-Sanders signature scheme:

bilinear map setting  $e: G_1 \times G_1 \rightarrow G_t$

secret key  $x, y$       public key  $X=g^x, Y=g^y$

signature on a message  $m$  is tuple  $(a,b)$  s.t.  $e(a,XY^m) = e(b,g)$

- with  $a$  random from  $G_1$
  - $b = a^{x+my}$

Leading to  $\text{PK}\{(a,b,m) : \quad 1 = e(a, XY^m) e(b,g)^{-1} \}$  ...however, this is not a valid statement :(

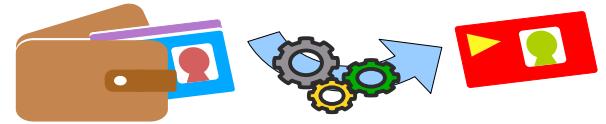


# Proof of knowledge of message and signature



So:  $\text{PK}\{(a,b,m) : e(a,XY^m) = e(b,g)\}$  } which, however, is not a valid statement :(

Trick:



1. Let  $(A, A') = (ag^r, h^r)$  i.e., encrypt  $a$  under



$$\rightarrow e(A, XY^m) = e(ag^r, XY^m) = e(b, g) e(g^r, XY^m) = e(b, g) e(g, XY^{rm})$$

$$\rightarrow e(A, X) = e(b, g) e(g, XY^{rm}) e(A, Y^{-m})$$

2. Let  $u = rm$

$$\rightarrow e(A, X) = e(b, g) e(g, XY^u) e(A, Y^{-m}) \wedge A' = h^r \wedge 1 = A'^{-m}h^u$$

Now we get

$$\text{PK}\{(u, r, b, m) : e(A, X) = e(b, g) e(g, XY^u) e(A, Y^{-m}) \wedge A' = h^r \wedge 1 = A'^{-m}h^u\}$$

which is a valid statement and proves that  $(Ag^{-r}, b)$  is a valid signature on  $m$  :)

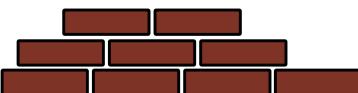


# Generalized Schnorr Proof Toolbox



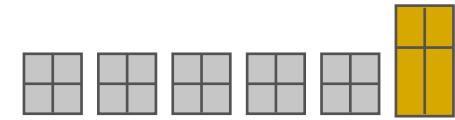
## Signature

Camenisch-Lysyanskaya (RSA, Pairings)  
Pointcheval-Sanders (Pairings)  
BBS+ Signatures (Pairings)



## Commitment

Pedersen  
ElGamal



## Encryption

Camenisch-Shoup  
Camenisch-Damgaard  
ElGamal



# Alternative construction frameworks

## Schnorr-proofs & discrete logarithms based primitives

- + well understood, lots of primitives, most efficient solutions
- non-interactive proofs under Fiat-Shamir heuristic

## Groth-Sahai proofs & structure-preserving primitives

- + well understood, fair amount of primitives, requires pairings (less efficient), short proofs
- security for many primitives rely on generic group model

## SNARKS, STARKS, bullet proofs & (algebraic) circuits

- + very expressive in terms of statements, short proofs
- much still in progress, need to design & research more primitives

## Lattice-based proofs & primitives

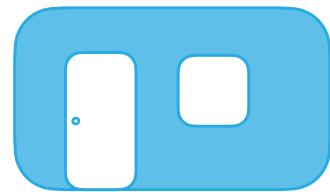
- + quantum safe, STARKS like proofs can be applied
- some non-interactive proofs require Fiat-Shamir heuristic
- very early stages, true quantum security ages away



## Part II

Difficulties in proving ZKP-using protocols UC-secure

# Cryptographic Protocol Design & Proofs

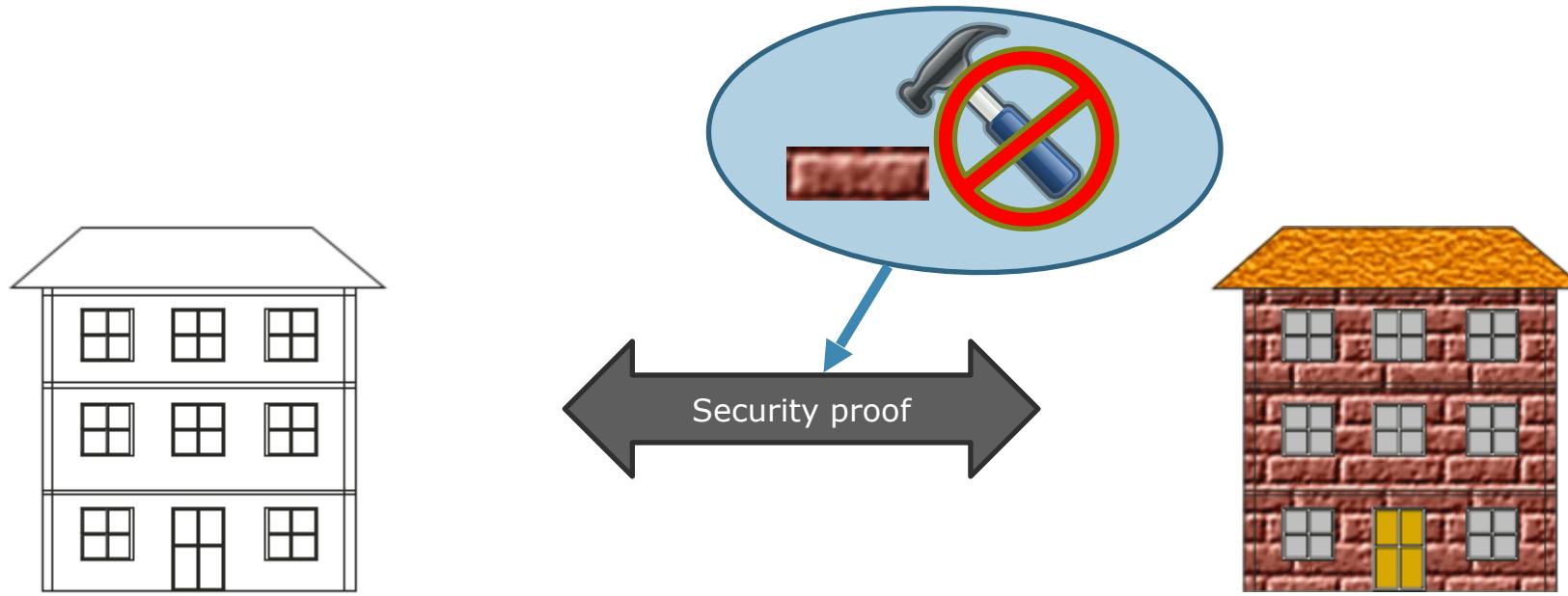


Specification:  
functional and  
security properties

Hybrid protocol

Actual implementation

# Cryptographic Protocol Design & Proofs

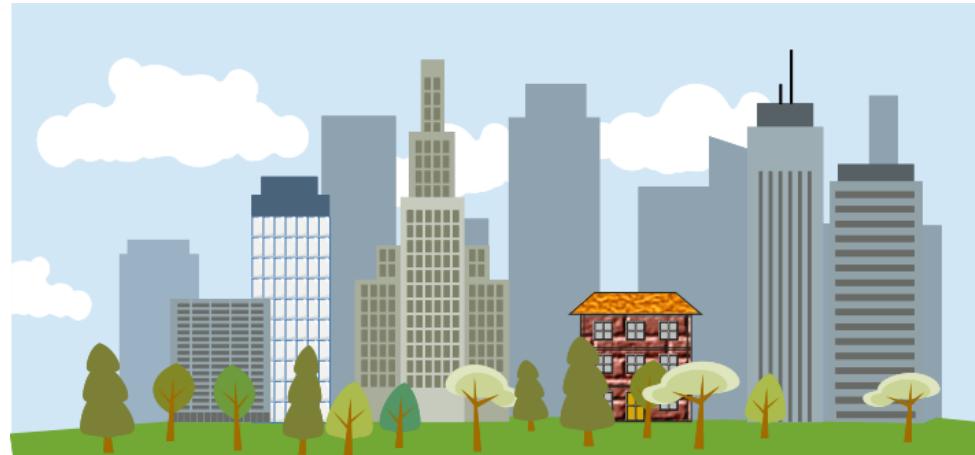
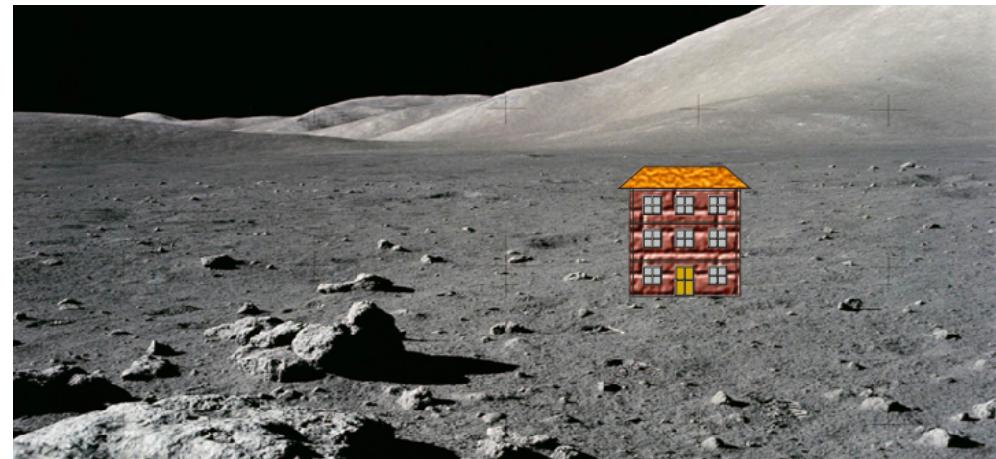
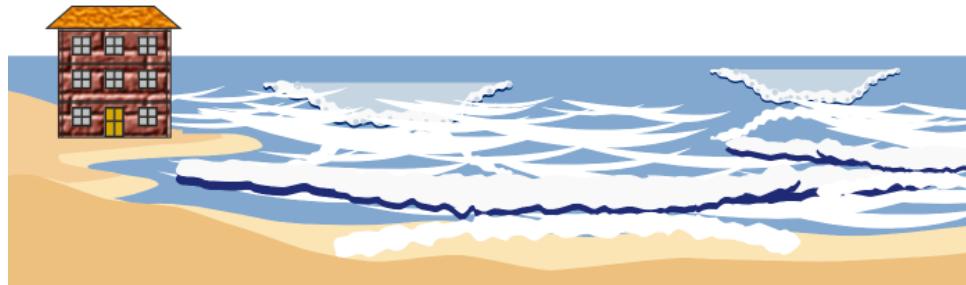


Specification:  
functional and security properties

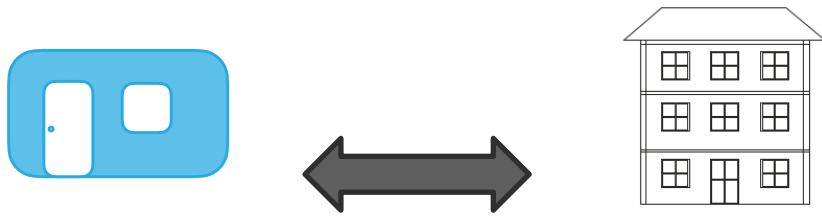
Actual implementation



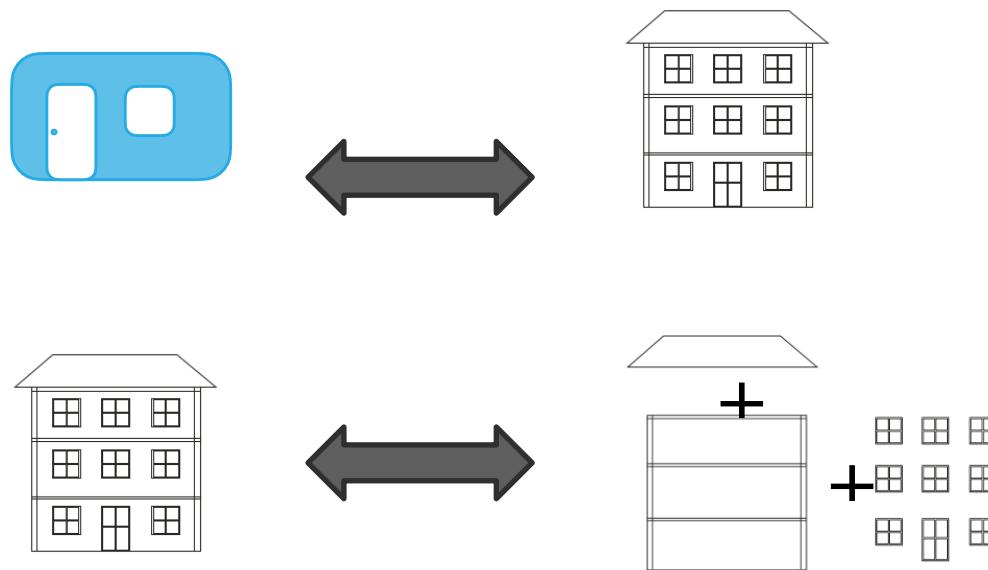
# Wanna guarantee security in *any* environment



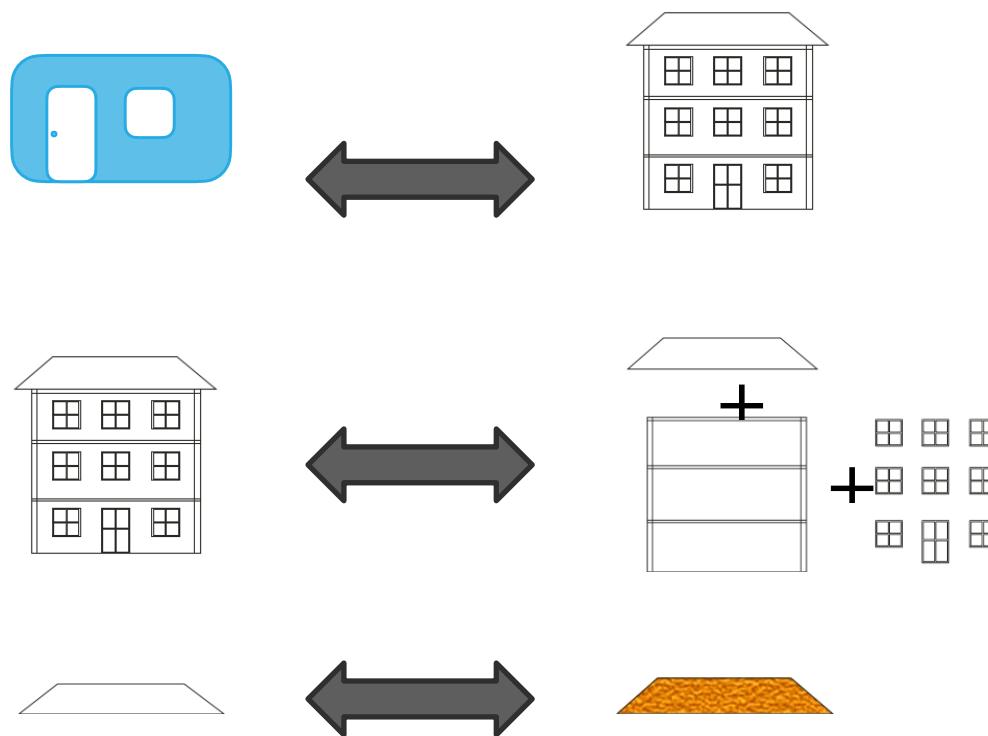
# Modular Cryptographic Protocol Design



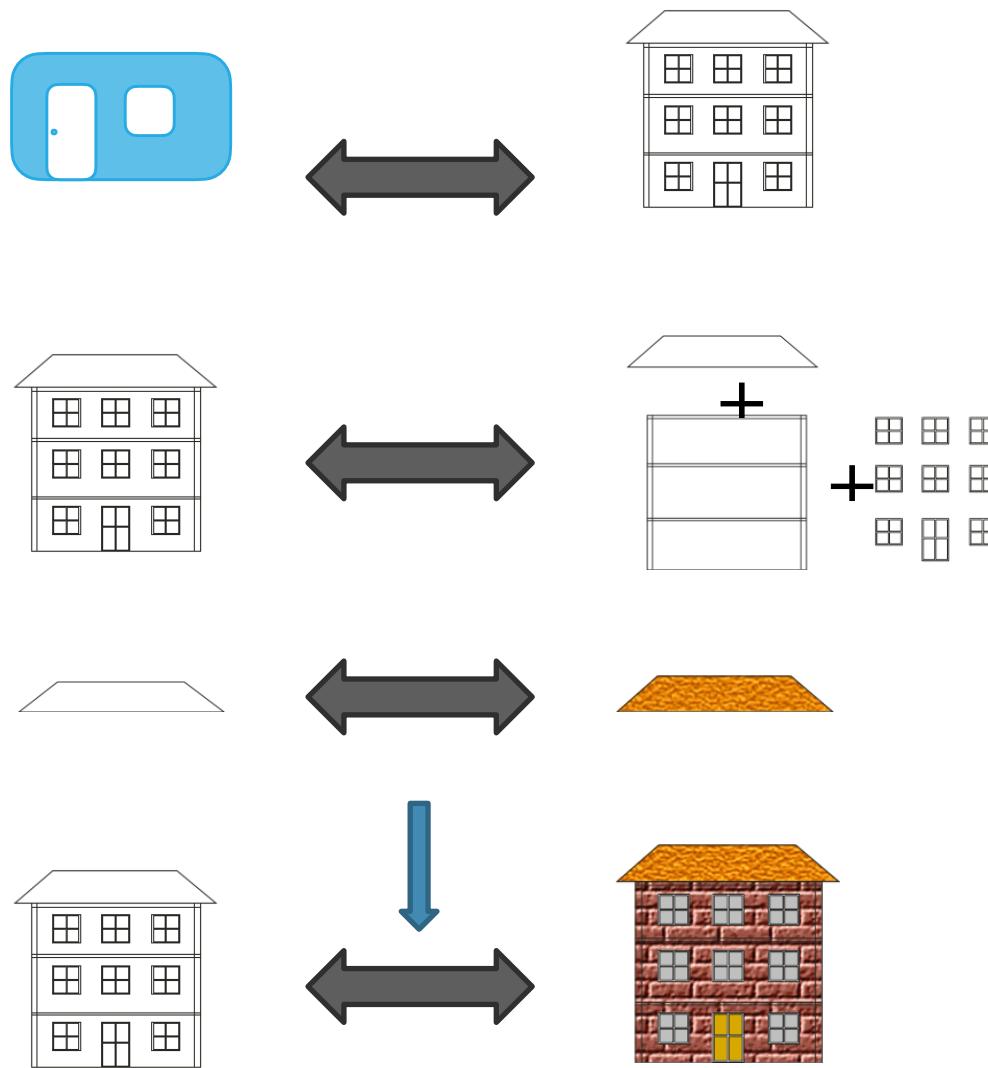
# Modular Cryptographic Protocol Design



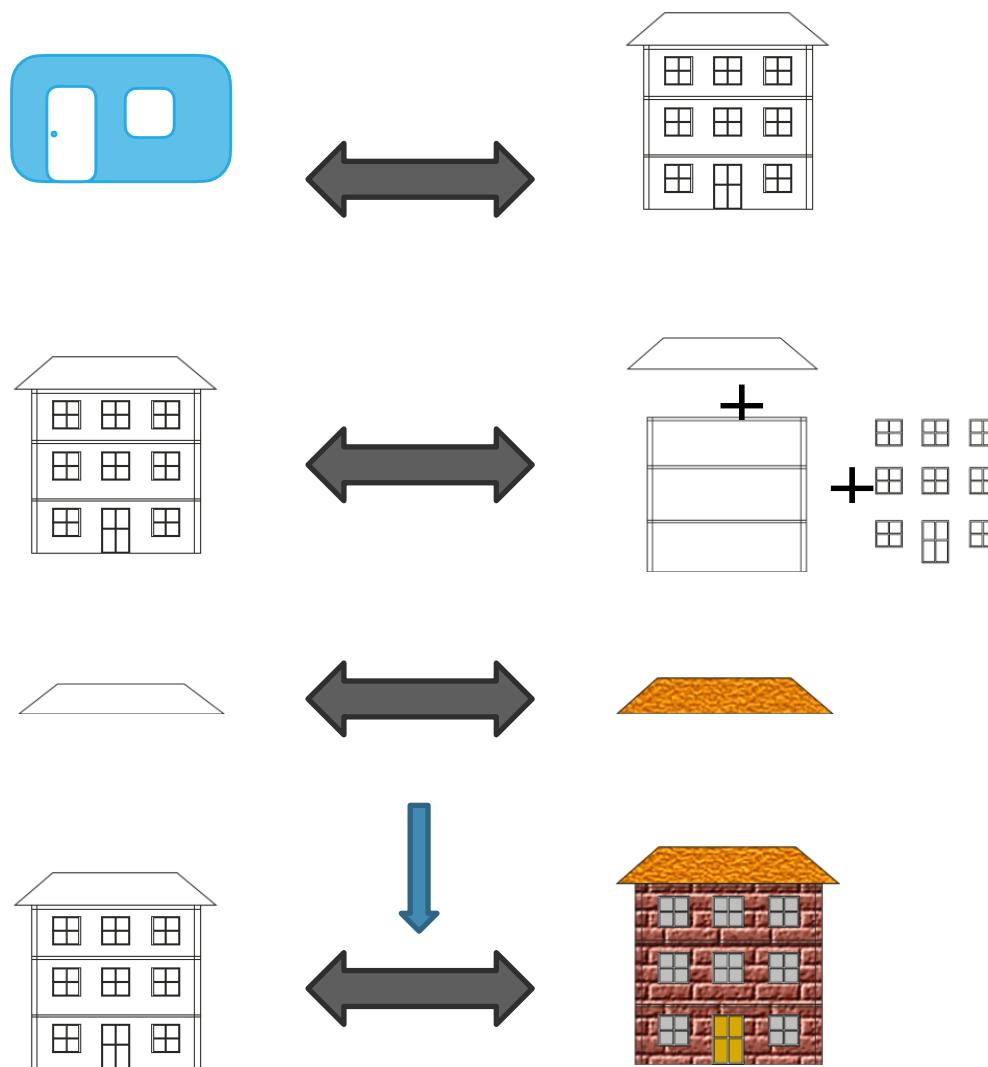
# Modular Cryptographic Protocol Design



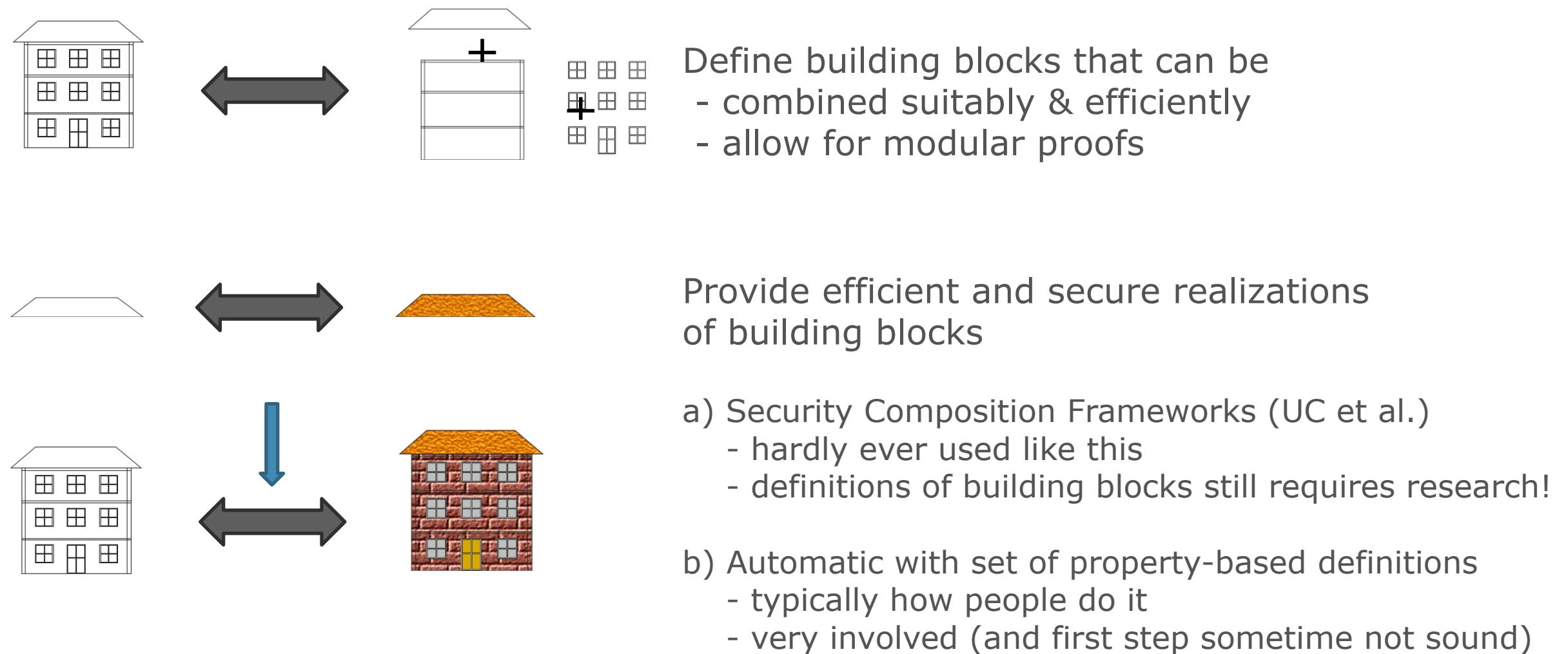
# Modular Cryptographic Protocol Design



# Modular Cryptographic Protocol Design



# State of the art & challenges

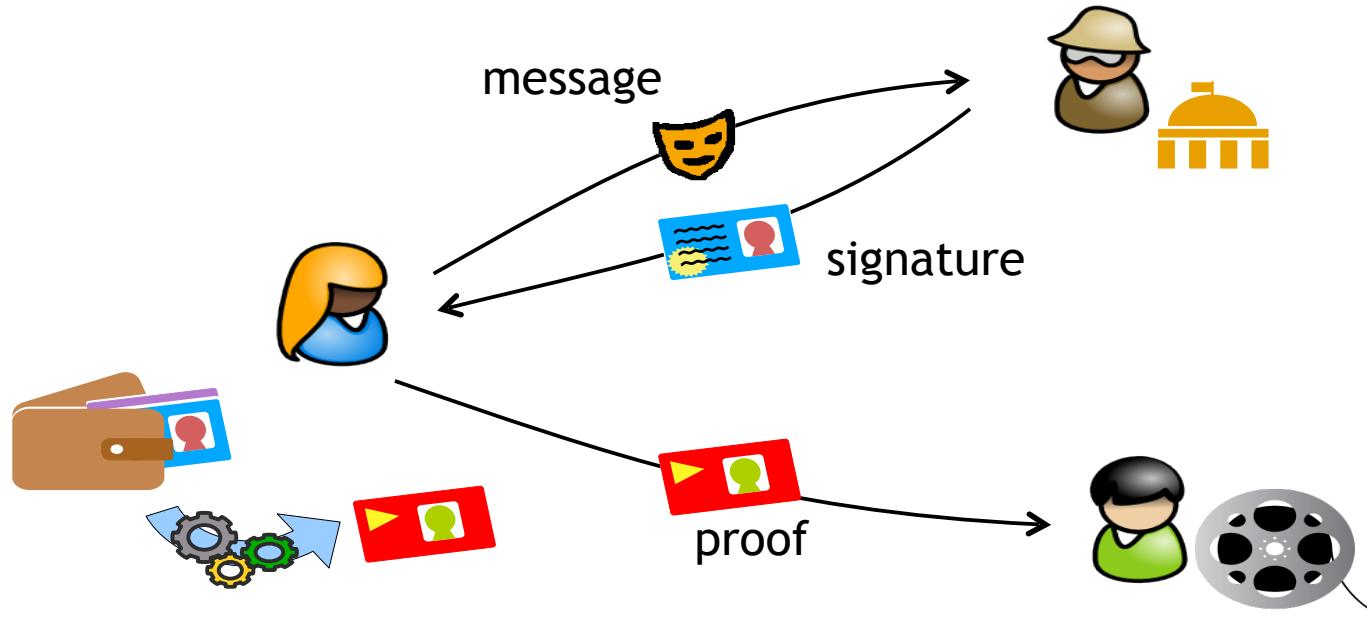


# Goals of (security) composition frameworks

- Modular construction with modular proof!
  - Functionalities are idealized
  - Protocol uses functionalities in a modular way
  - As protocol is modular, proof is be modular and, in particular, abstract and simple
  - As functionalities model the security of the schemes, no additional reduction to any property of the schemes should be necessary!



# Consider (very) simple anonymous credential use case



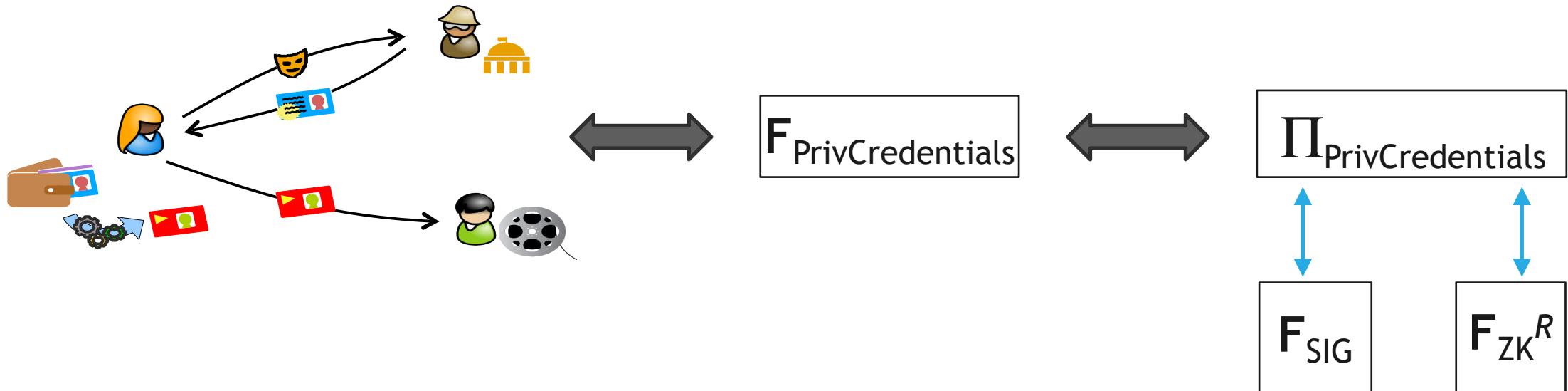
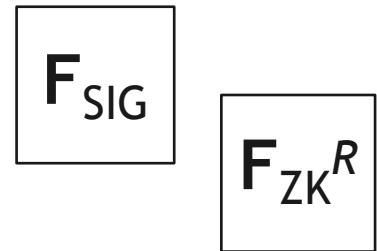
proof:  $\text{PK}\{(s,m): \quad 1 = \text{verify\_signature}(\text{PK}, s, m)\}$

# Let us UC-dream....

# modular protocol construction

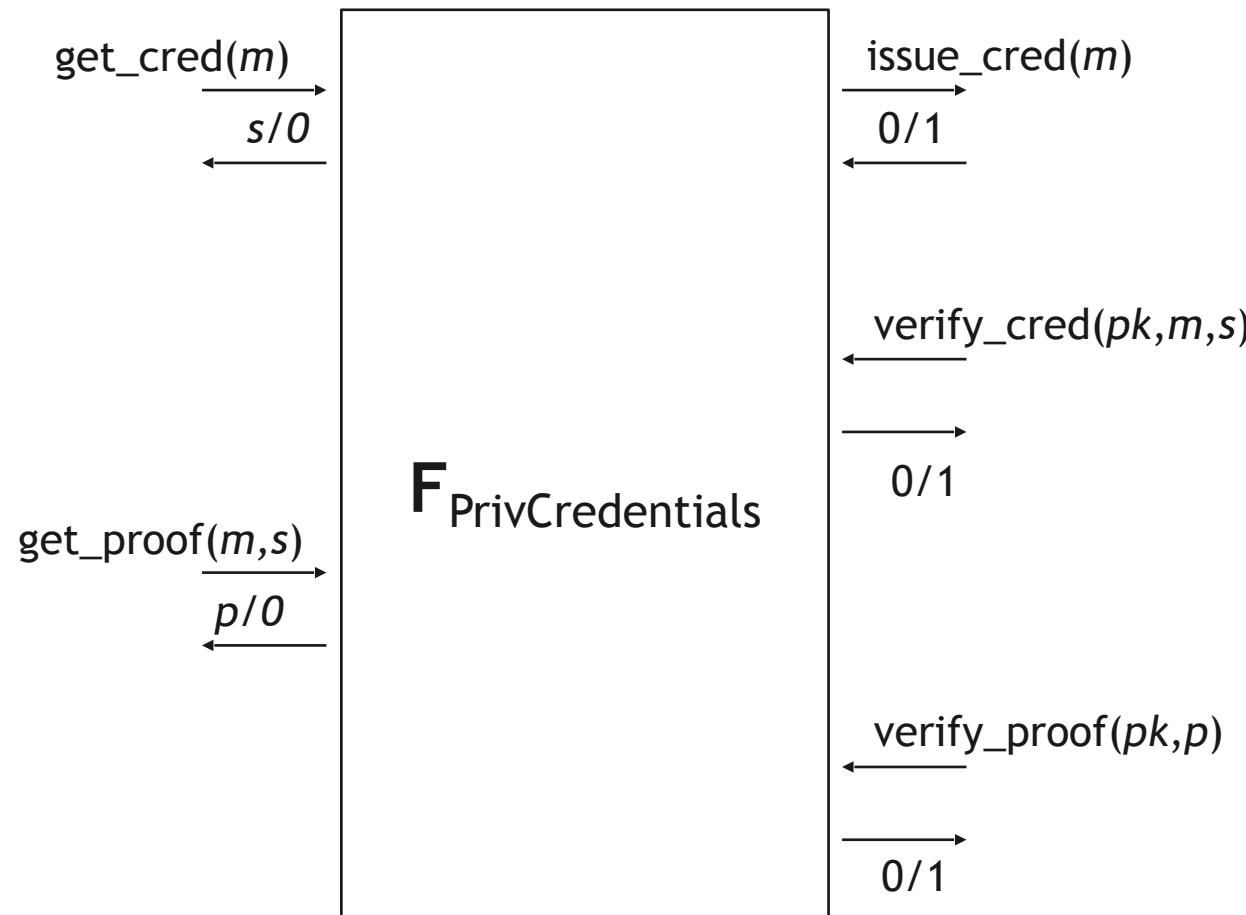
UC building blocks are available!

- signature scheme functionality  $F_{SIG}$  (e.g., Canetti '04)
- zero-knowledge proof functionality  $F_{ZK}$  (e.g., Canetti UC'05)



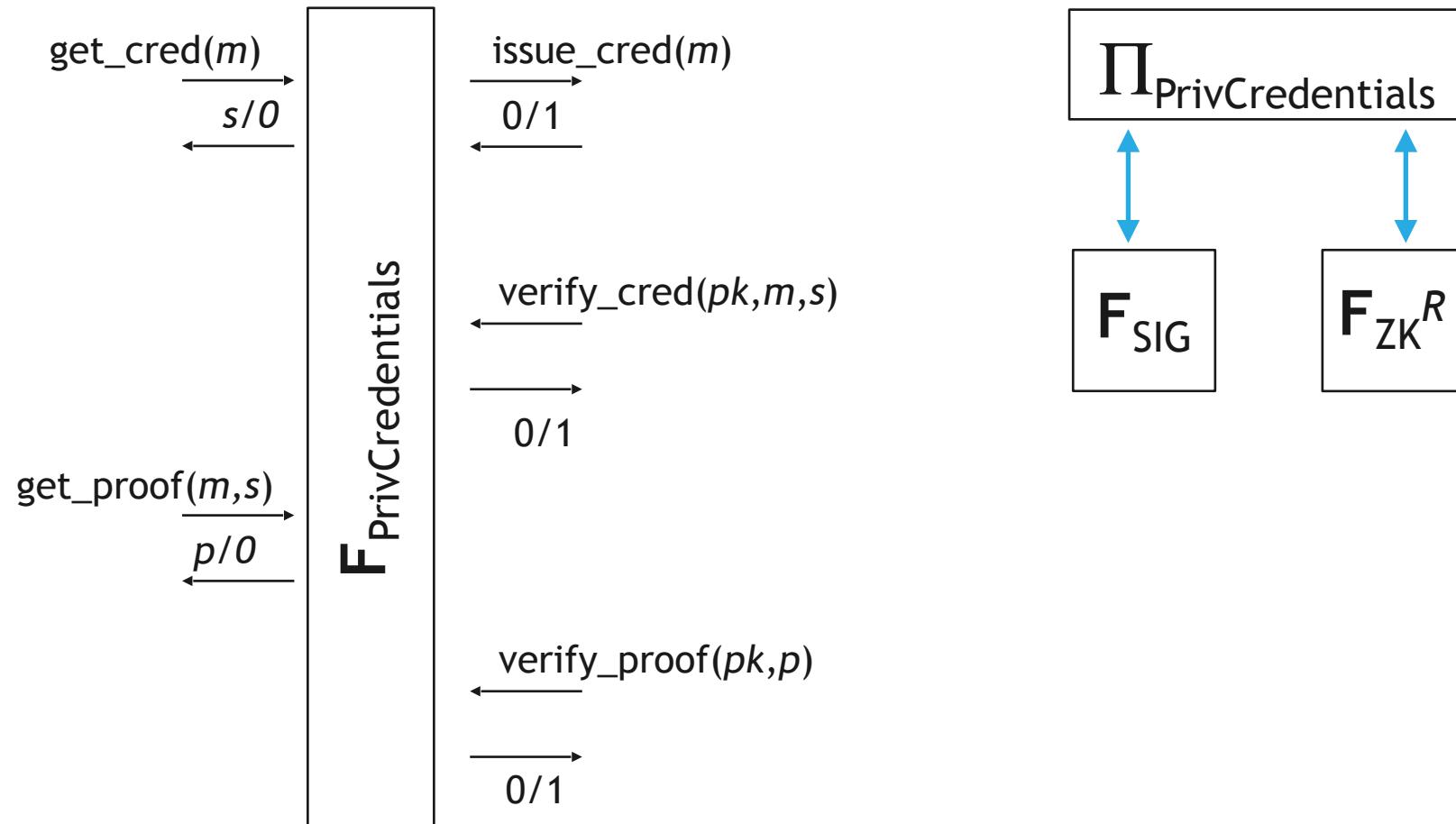
# Let us UC-dream....

# modular protocol construction



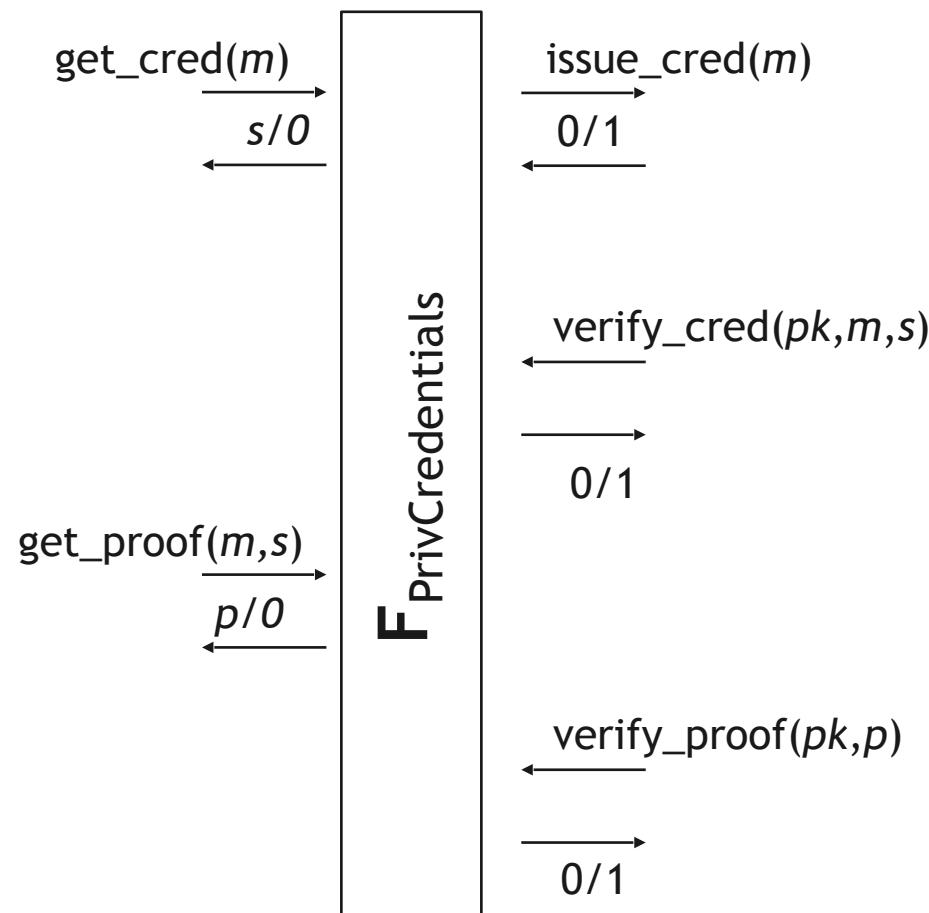
# Let us UC-dream....

# modular protocol construction



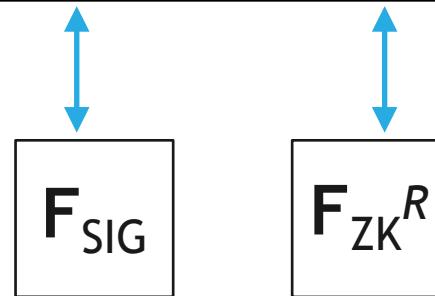
# Let us UC-dream....

# modular protocol construction



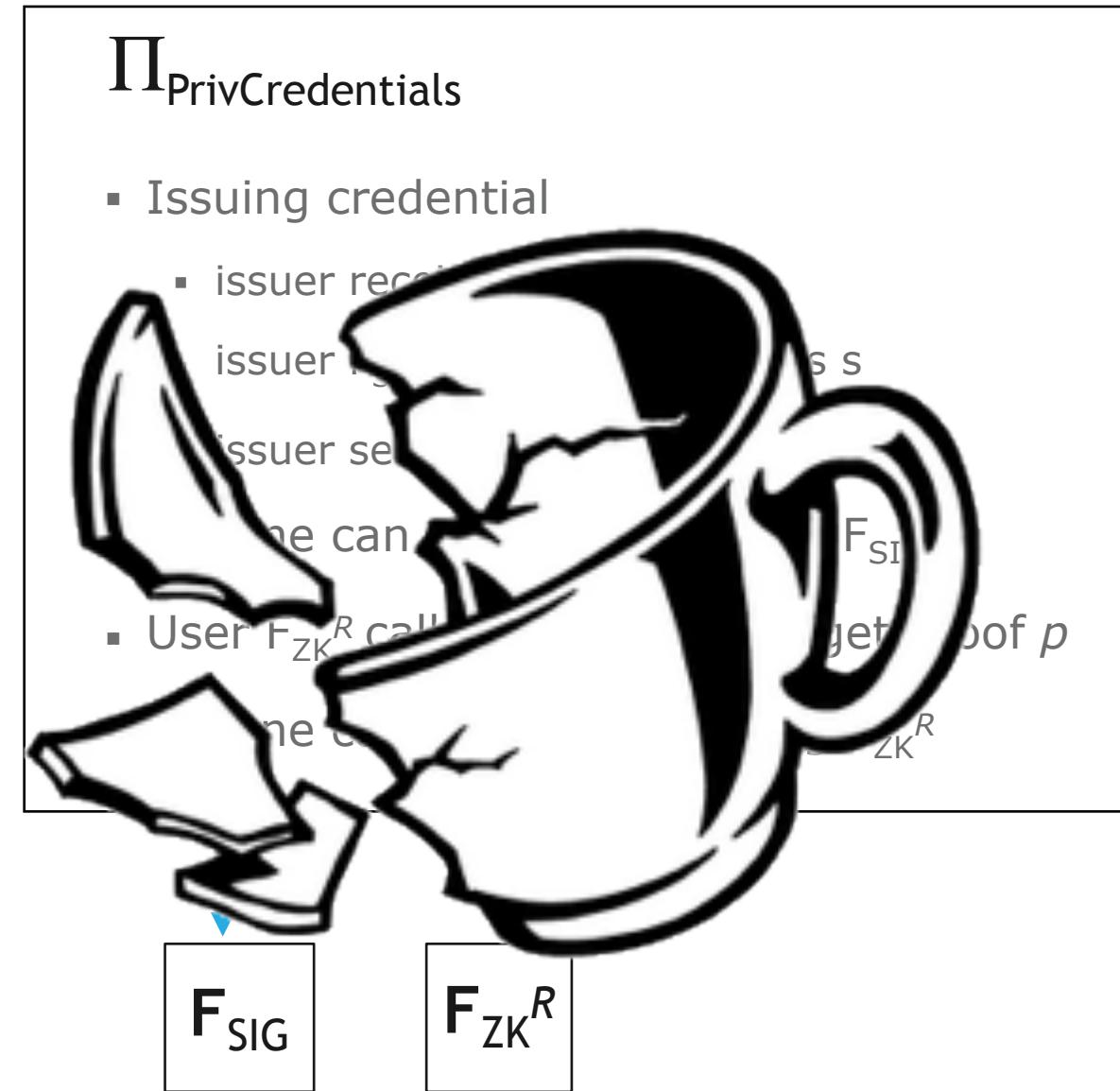
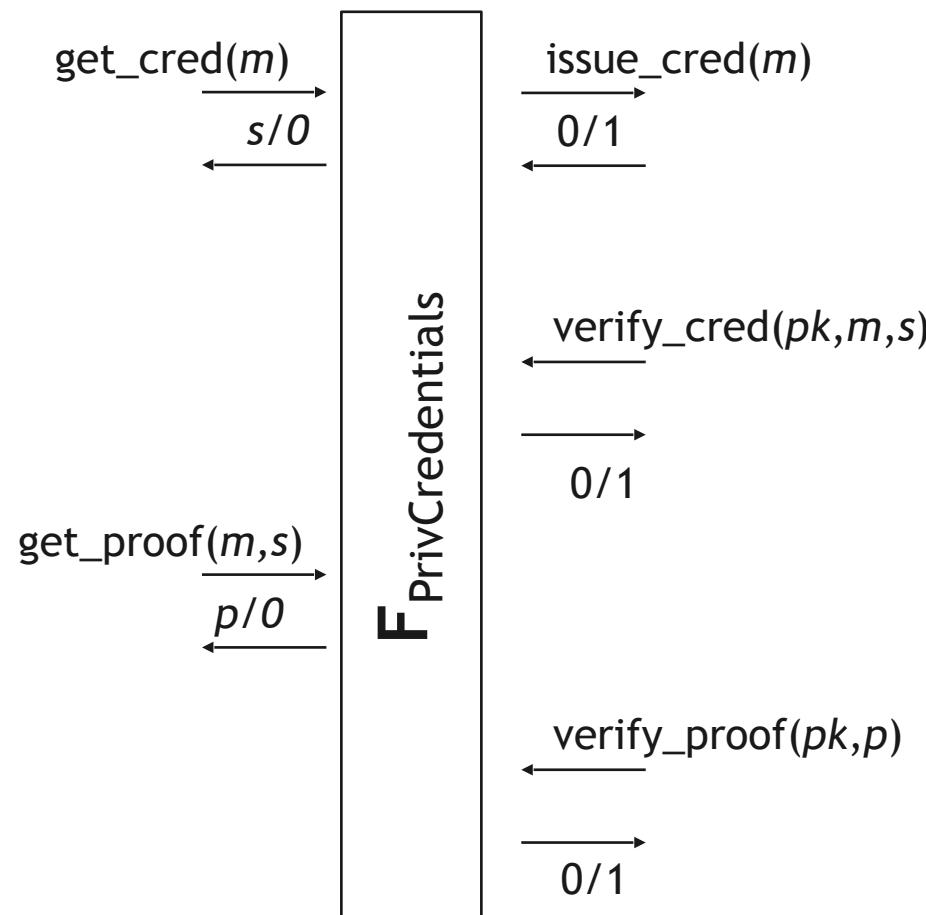
$\Pi_{\text{PrivCredentials}}$

- Issuing credential
  - issuer receive  $m$  from user
  - issuer  $F_{\text{SIG}}$  calls on  $m$  and gets  $s$
  - issuer send  $s$  to user
- Anyone can verify  $s$  by calling  $F_{\text{SIG}}$
- User  $F_{\text{ZK}}^R$  calls on  $m, s, pk$  to get proof  $p$
- Anyone can verify  $p$  by calling  $F_{\text{ZK}}^R$



# Let us UC-dream....

# modular protocol construction



# Let us investigate our building blocks



Different versions of  $F_{SIG}$  exist:

- Every signature generated by adversary (e.g., Canetti ‘04)
- Adversary initially uploads algorithms (e.g., Canetti UC’05, Küsters-Tuengerthal ‘13)

The relation  $R$  is between statement  $y$  and witness  $w$  and parametrizes the functionality. The proof is of course only accepted if  $(y, w)$  is in the relation  $R$ , and the functionality models a proof of knowledge.

$(y, w)$  in  $R$  if  $y = (pk, crs)$  and  $w = (m, s)$  and  $verify(pk, m, s) = 1$



# Let us investigate our building blocks



A number of stumbling stones:

- $F_{\text{ZK}}^R$  is interactive, but we need to get a “proof” string
- Relation  $R$  talks about a specific algorithm, i.e.,  $\text{verify}(pk, m, s) = 1$ , but  $F_{\text{SIG}}$  is agnostic of its algorithms, i.e., defined for *any* algorithms
- The statement  $\text{verify}(pk, m, s) = 1$  does not imply that  $F_{\text{SIG}}$  would answer with 1



# Towards removing the stumbling stones

A number of stumbling stones:

- $F_{ZK}^R$  is interactive, but need to get a “proof” string:
  - Use (modification of)  $F_{NIZK}^R$  that outputs a proof [Groth et al 2012]
- Relation  $R$  talks about a specific algorithm, i.e.,  $\text{verify}(pk, m, s) = 1$ , but  $F_{SIG}$  is agnostic of its algorithms, i.e., defined for *any* algorithms
  - Change  $F_{SIG}$  to be parameterize by algorithms (otherwise follow [Can05, KT13])
- Still, the statement  $\text{verify}(pk, m, s) = 1$  does **not** imply that  $F_{SIG}$  would answer with 1
  - Non-trivial.... and is not what we want!

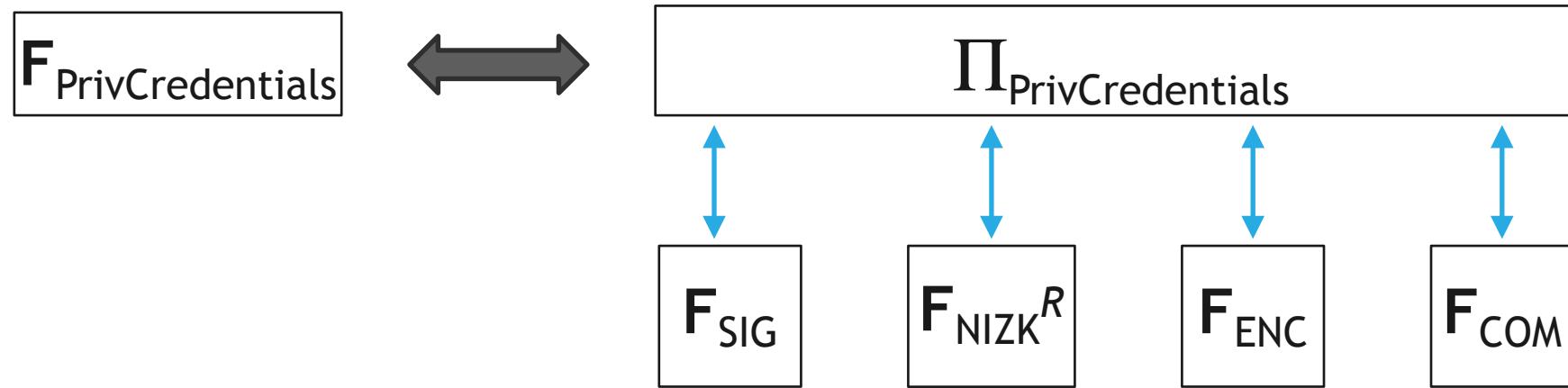


# Part III

Modular ZK protocol usage  
What we want and how to do it

Let us UC-dream....

modular protocol construction

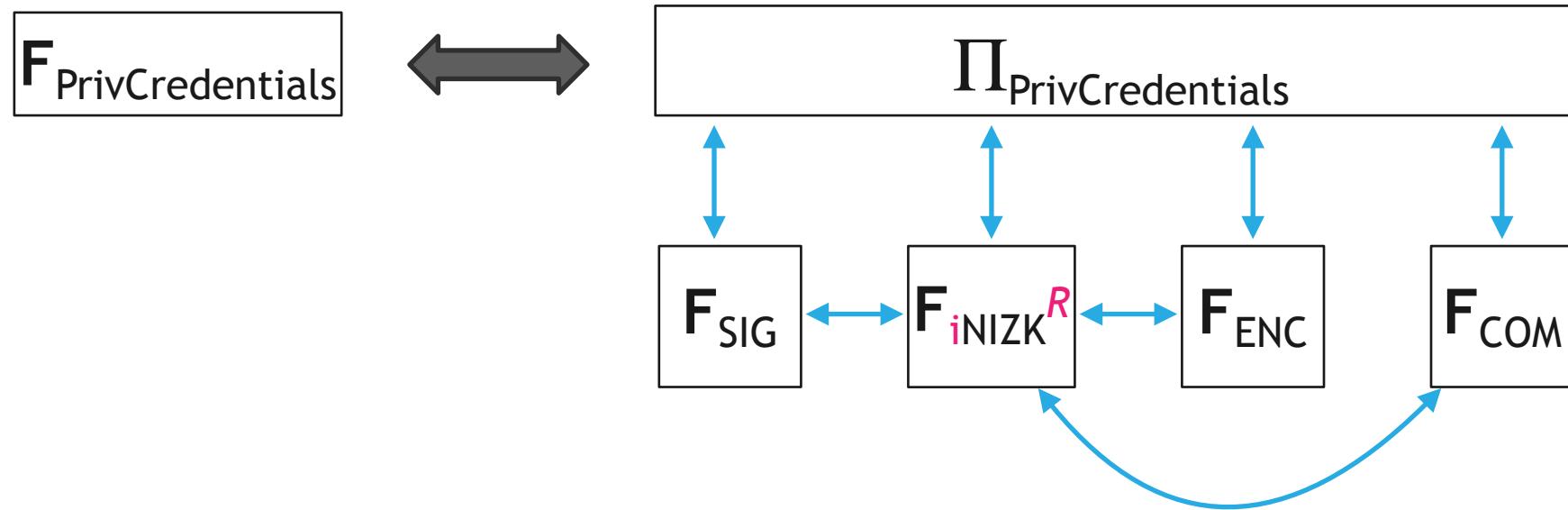


with relation R making statements about results of algorithms  
(e.g., verify, encrypt, and open) used in other functionality



Let us UC-dream....

modular protocol construction

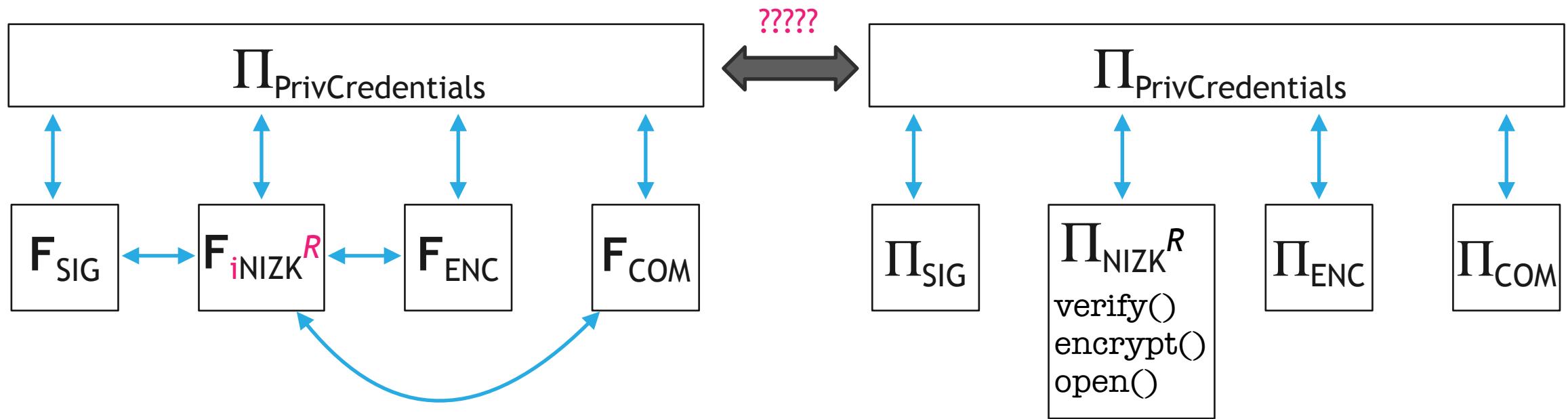


with relation  $R$  making statements about results of **calls** to other functionalities!



# Let us UC-dream....

# modular protocol construction

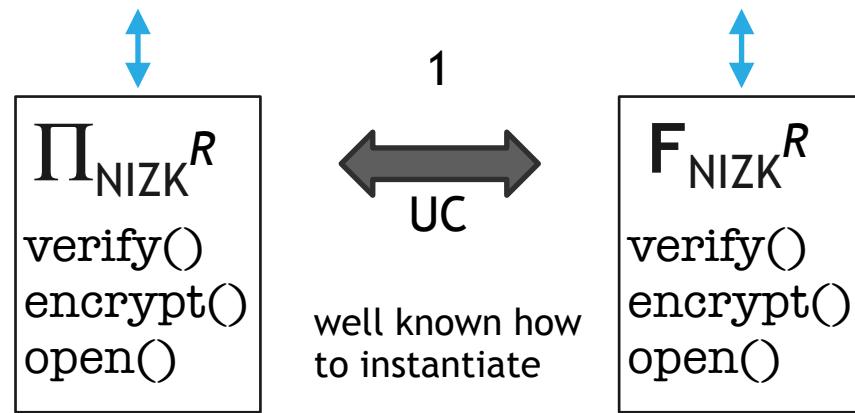


Two issues:

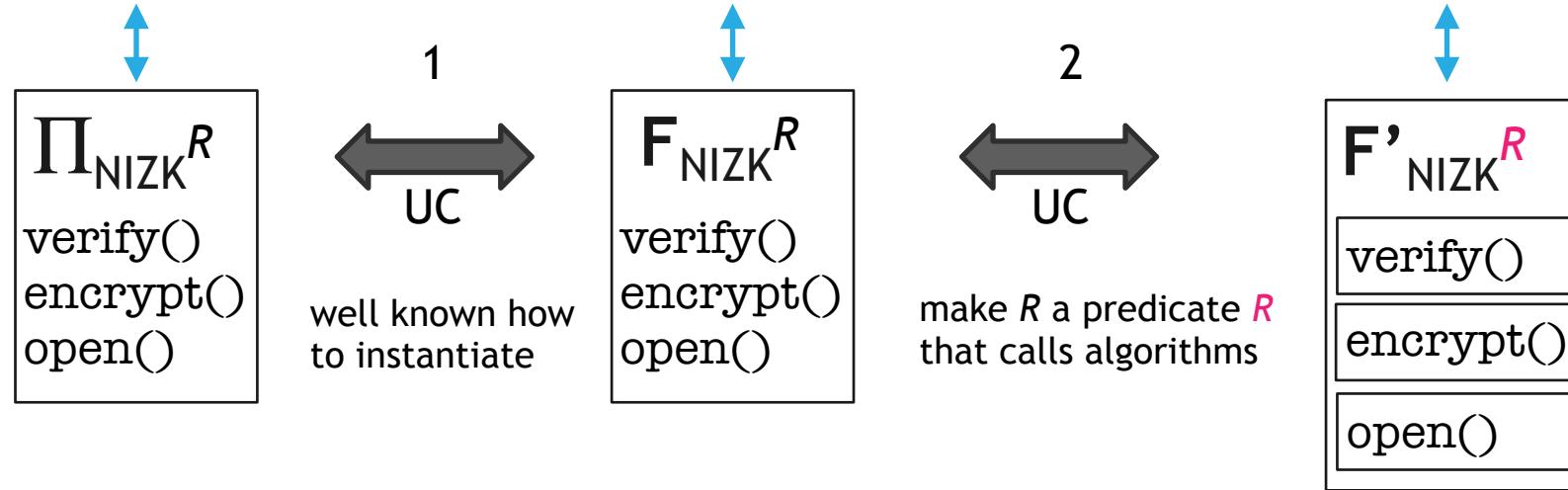
- $\Pi_{PrivCredentials}$  is not subroutine respective, i.e., UC theorem does not apply !
- $\Pi_{NIZK}^R$  does not interact with other protocols and thus cannot realise  $F_{iNIZK}^R$  !



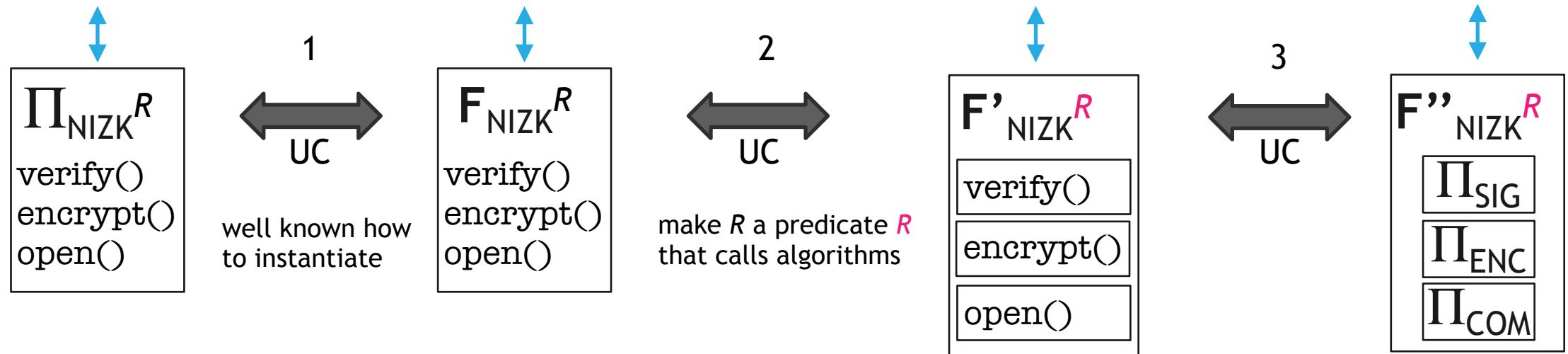
# Composable modular protocol construction, finally!



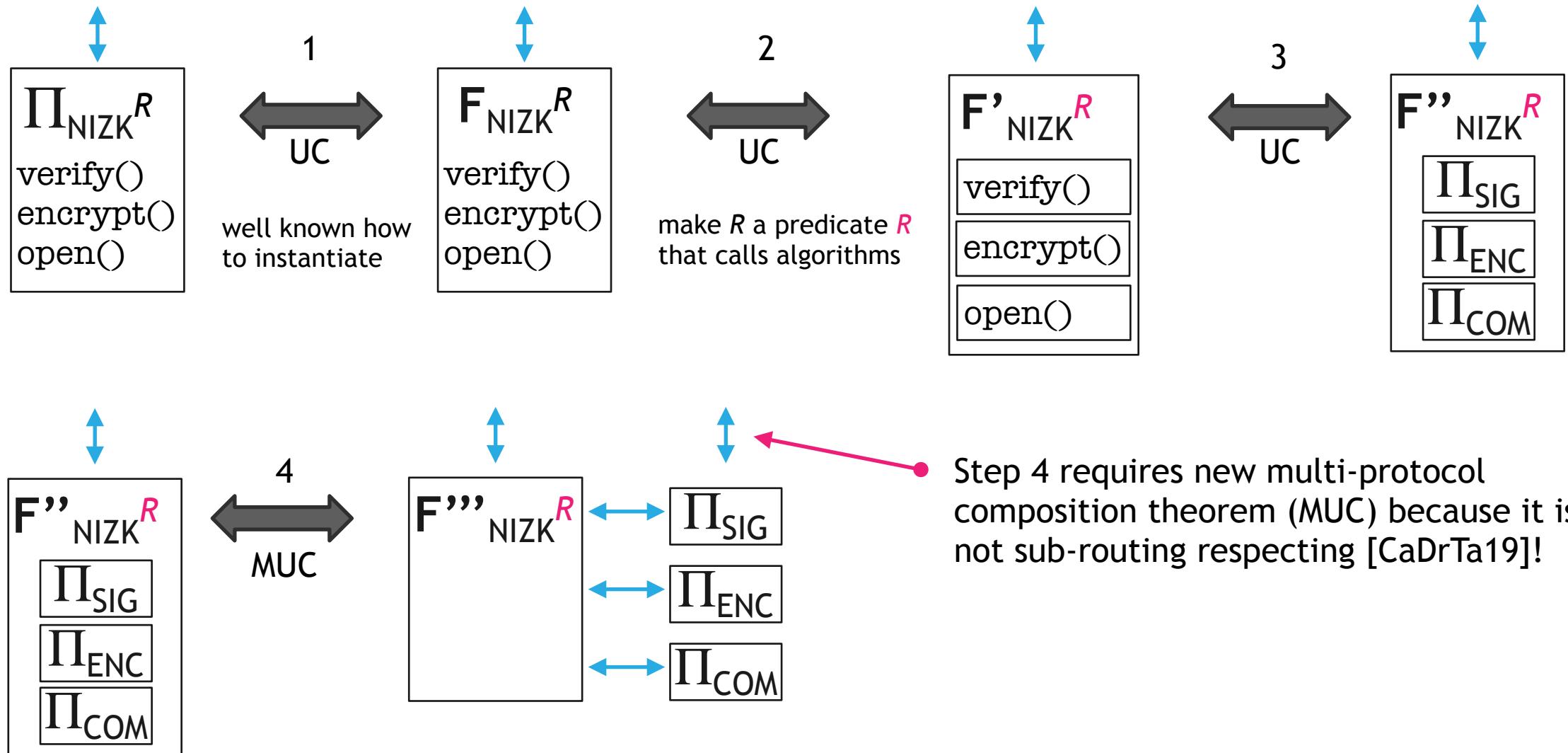
# Composable modular protocol construction, finally!



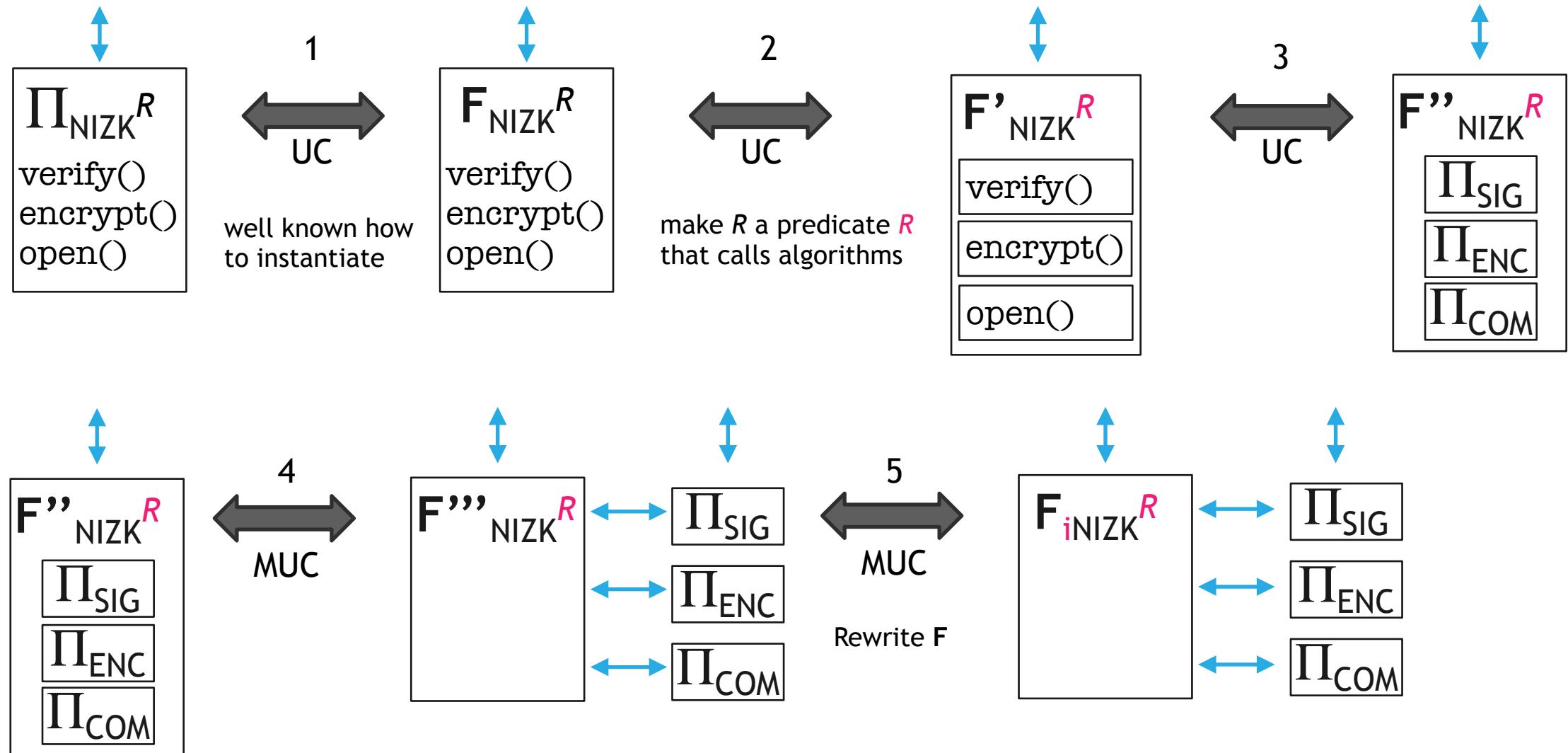
# Composable modular protocol construction, finally!



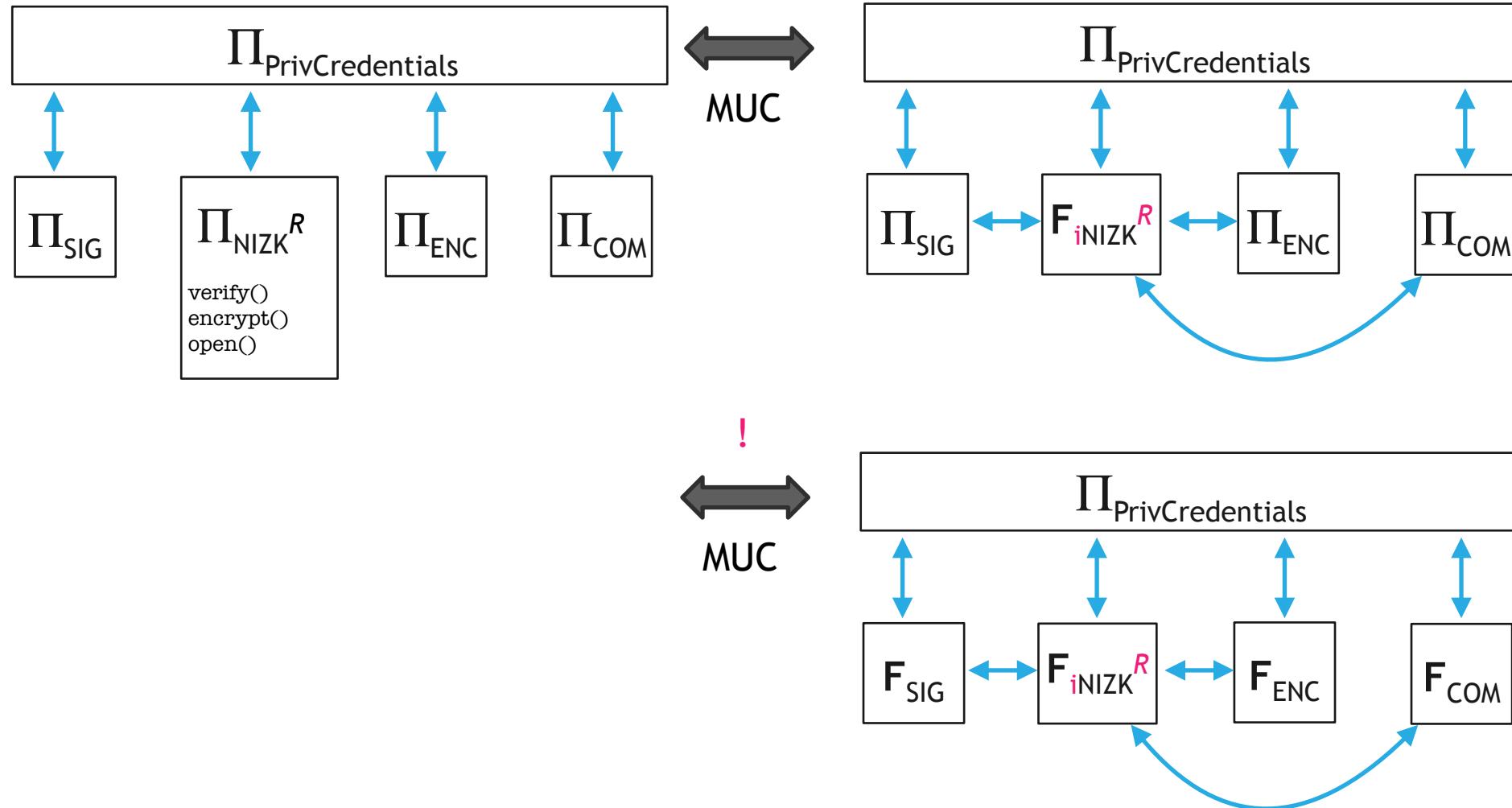
# Composable modular protocol construction, finally!



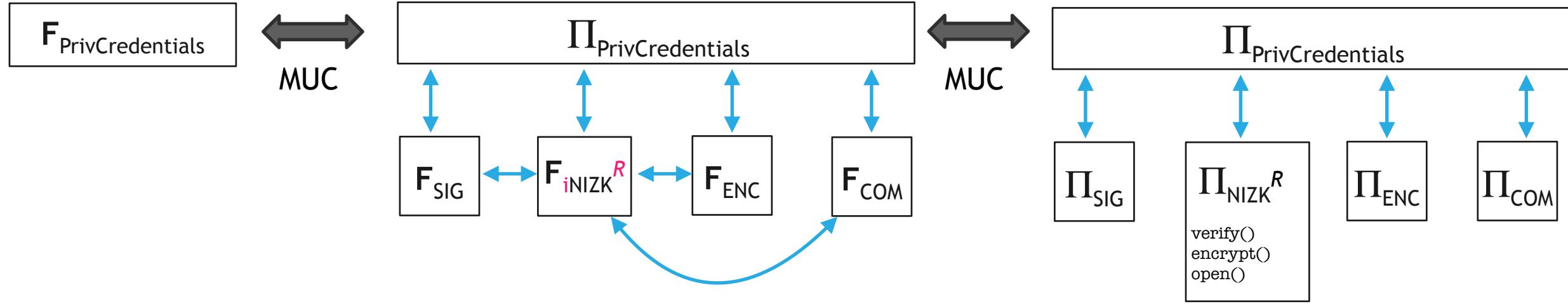
# Composable modular protocol construction, finally!



# Composable modular protocol construction, finally!



# Composable modular protocol construction, finally!



Summary of proof method:

- proof needs to be done for each predicate  $R$
- proof is generic, i.e., works for any instantiations, except Step 1 (realisation  $\Pi_{\text{iNIZK}}^R$  of  $F_{\text{iNIZK}}^R$ )

Protocol design:

- 1) define overall  $F$
- 2) define  $\Pi$  and  $F_{\text{iNIZK}}^R$
- 3) prove security of  $\Pi$
- 4) apply steps above!

Details see [Camenisch, Drijvers, Tackmann '19] <https://ia.cr/2019/065> (also shows concrete examples)



# Conclusion

## Part I

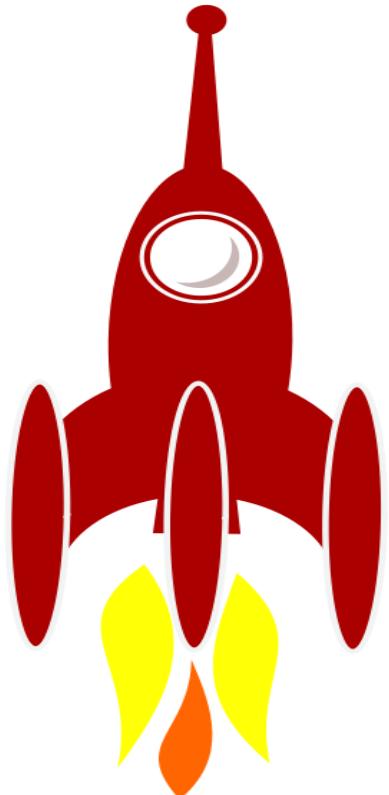
- A number of ZKP frameworks
- Generalised Schnorr proofs ready to be standardised
- More work is needed for post-quantum frameworks

## Part II

- The UC world is not as we would think it is...

## Part III

- Modular proofs are very important!
- UC Functionalities in the literature are typically not suited for composability
- UC Theorem is too limited -> MUC Theorem
- New proof methodology for modular zero-knowledge functionality
  - Needs to be done per relation
  - But otherwise works for any realisation of functionality
- More work is needed before we can become space farers!





# Thank you!

[ia.cr/2019/065](https://ia.cr/2019/065)