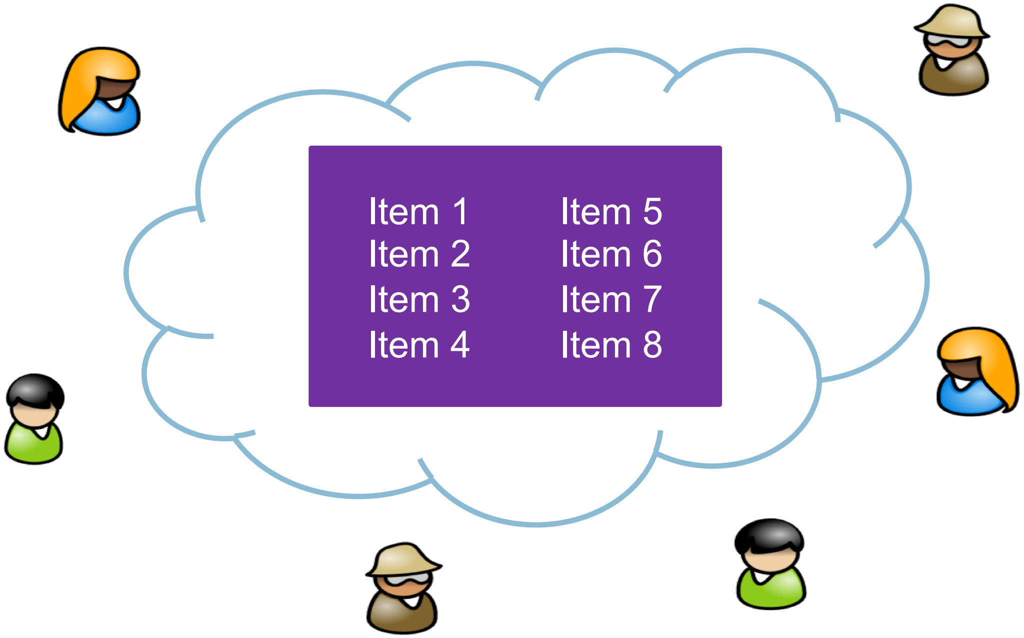
# Cryptography in Crypto

Jan Camenisch
Head of Research

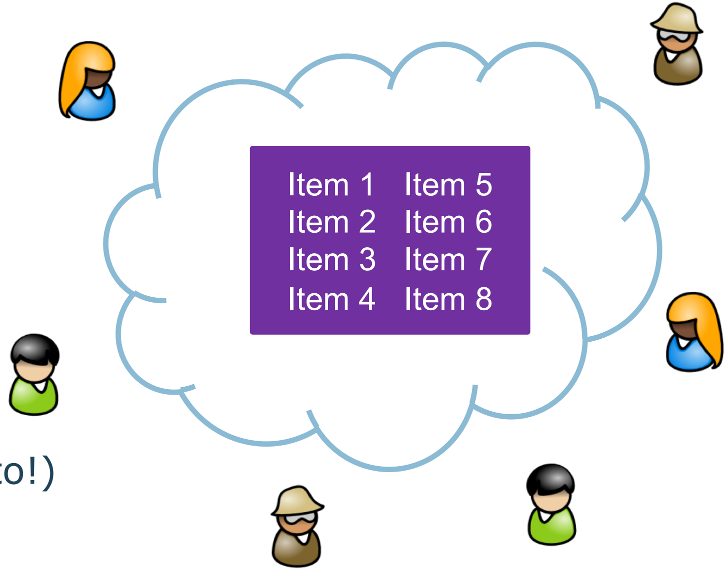
@janCamenisch jan.camenisch.org jan@dfinity.org

# A distributed bulletin board




No trust in a single party, but in a majority, so less trust needed

# Distributed Bulletin Board




Problem: Items are sent as input to all parties, but possibly are received in different order! Thus, parties need to agree on inputs and their order. How?

Millennial solution:

- Select a leader (lottery – different interesting crypto!)
- Leader makes proposal
- Parties sign proposal if they agree with leader
- Full agreement if >1/2 (or >2/3) signatures
- If no agreement start over (no proposal or insufficient sigs)

Not quite solved:

- Might have to start over quite a few times, so not really practical?
- Who are the parties who participate?
- Running a distributed protocol often very costly

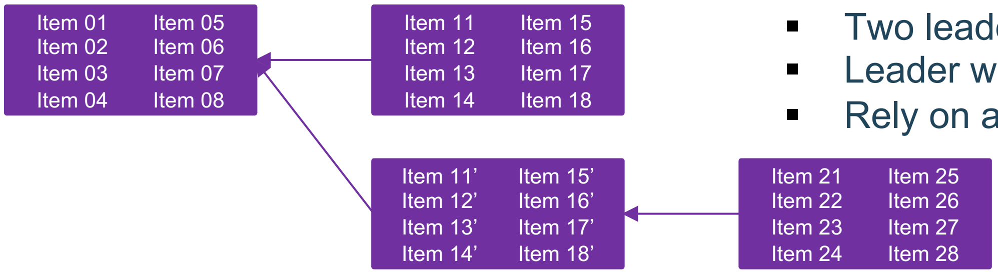
# Bitcoin blockchain solves this

1. Public lottery
    1. Need to limit amounts of times single individual can participate
    2. Anyone with sufficient computational power is allowed to participate
    3. Find pre-image of hash-function s.t. output has last x bits = 0

2. Combine lottery with authentication of proposal by winner (selected leader):

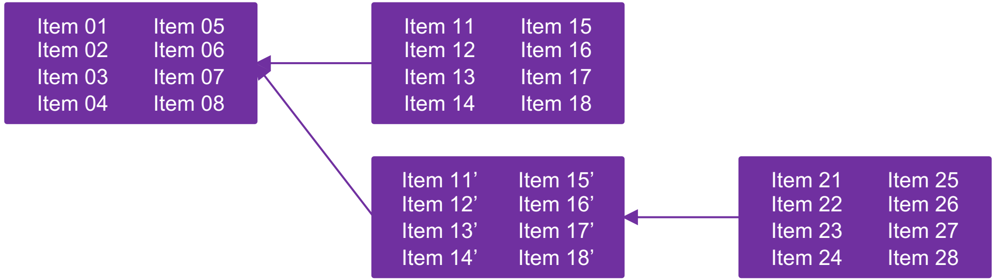
    Hash(item1, … , item8, random) = 0x****000000

3. Eventual agreement, i.e., allow temporary disagreement (forks)




- Two leaders made a proposal each
- Leader was malicious and made two proposals
- Rely on agreement being found at some point

# Bitcoin blockchain continued




Interesting crypto problems:

1. Prove the security of this construction (see literature for more)
   - what does it achieve?
   - under what assumptions?
   - under what adversarial models?

2. Huge drawback: uses way to much computational power, can we do better?

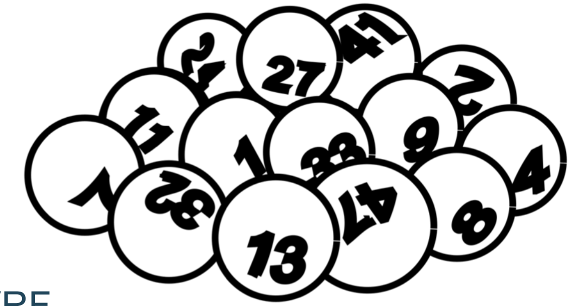
# More Interesting Crypto Problems

How do we do a lottery?

Use a (pseudo) random function to select leader (i.e., list of ranked leaders):

a. Global random function (random beacon)
   - requires multi-party computation
   - leader is known to all, potentially vulnerable to adaptive attacks
   - only top ranked leaders need to act

b. Local random function
   - parties need be able to prove they executed function correctly: VRF
   - leader only known, if all parties have announced their results
   - protects better against adaptive attacks



# Global random function (random beacon)

Requirements: threshold verifiable (pseudo)random function

- Regularly provide fresh pseudo random (as soon as >1/2 or >2/3 decide new period has started)
- Efficient computable by distributed protocol
- Provably secure

$\mathcal{F}_{\text{tVRF}}(\tau, \gamma)$

new($P_j$, t) →

rand(t) →

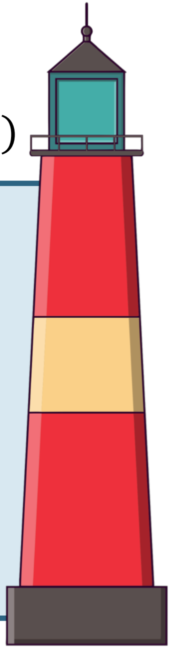
← r

$(P_1, \ldots, P_n)$, $k = 0$, $S_i = \{\}$

If $P_j \in (P_1, \ldots, P_n)$ and $t = k+1$ then $= S_t \cup \{P_j\}$

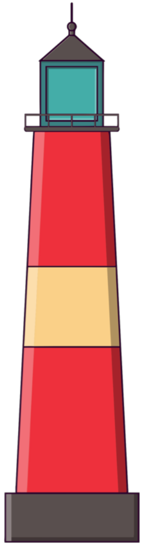
If $|S_t| = \tau$ then $r_t = \text{random}(\gamma)$

If $0 < t < k+1$ then $r = r_t$ otherwise $r = \perp$



[Cachin, Kursawe, Shoup ’00, Hanke, Movahedi, Williams ‘18]

# Realization of random beacon

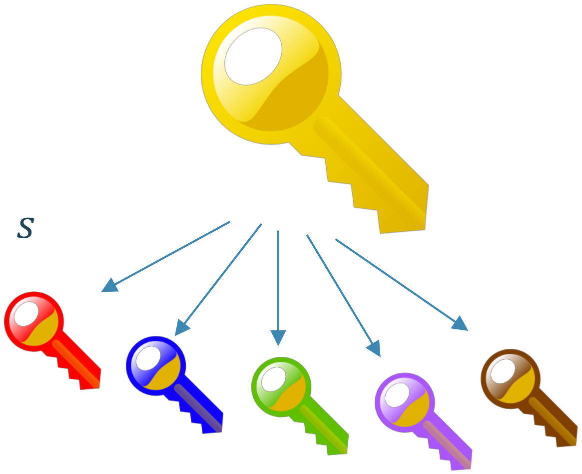


Idea: use *non-interactive* threshold signature scheme

- Threshold:
  - n parties; single public key $y$, each party holding a secret key share $x_i$
  - if at least $\tau$ provide partial signature $s_i$ on $t$, reconstruct sig $s$
- Non-interactive: parties can generate & publish partial signature
- Signature scheme:
  - $r_t = Hash(s) = Hash(sig_x(t))$ is random in the random oracle model if
    - signature is unique *(not usually the case)*
    - signature is unpredictable *(implied by unforgeability)*
  - Only know candidates for efficient scheme are RSA and BLS

[Shamir 79]

# Shamir's Secret Sharing



Share a secret $s$ among n parties with threshold $\tau$:

- Define a random polynomial $p(.)$ of degree $\tau$, s.t. $p(0) = s$
- Let share $s_i$ for party $P_i$ be $s_i = p(i)$

Security properties:

- Any set $\{s_i\}$ of less than $\tau$ shares contain *no information* about $s$
- Given a set $\{s_i\}$ of $\tau$ or more shares, secret $s$ can be reconstructed (interpolation)
  - $s = \sum_i \lambda_i s_i$ (where $\lambda_i$ are public coefficient depending on the particular set)

Notice: interpolation is a linear in shares!

[Boneh Lynn Shacham ‘04]

# BLS Threshold signature scheme

## BLS signature scheme

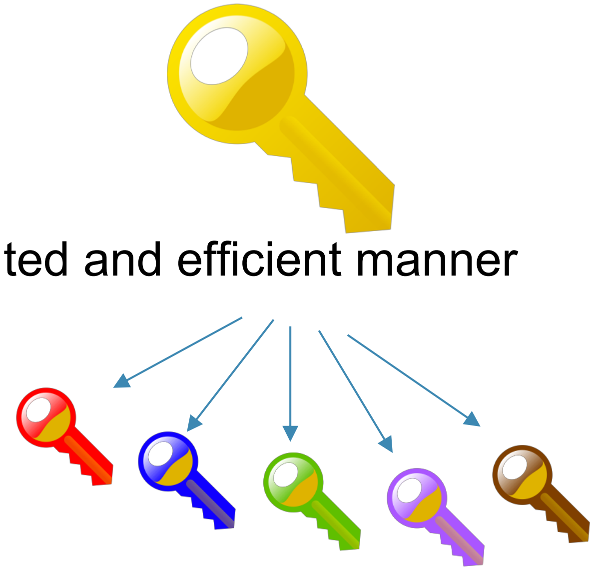
- Works in an algebraic group with generator $g$ of order $q$
- Secret key: random $x \in \mathbb{Z}_q$, Public key: $y = g^x$
- Signature on message $s = Hash(m)^x$

## Threshold BLS

- Assume secret key $x$ is shared: $x_i = p(i)$, partial public key $y$
- Signature share $s_i = Hash(m)^{xi}$
- Signature reconstruction from signature shares

$$s = \Pi \, s_i^{\lambda i} = Hash(m)^{\Sigma \, \lambda i \, xi} = Hash(m)^x$$

# Distributed Key Generation



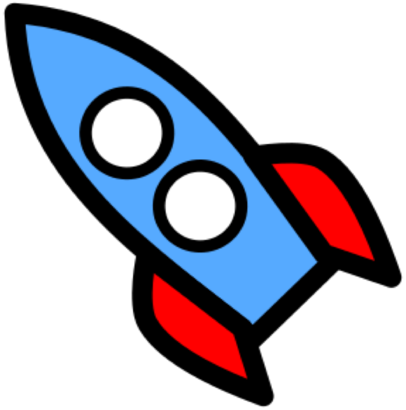
- Need to generate public keys and shares in a distributed and efficient manner
- Notice: Shamir's secret sharing is linear:
  - Let $p1()$ and $p2()$ share $s1$ and $s2$, respectively
  - Then $p() = p1() + p2()$ shares $s = s1 + s2$
- Thus, we can implement DKG by
  - Set of dealers each sharing random value & use NIZK that they did this correctly
  - Agree on dealers with correct NIZK (using blockchain ☺)
  - Locally sum up shares received from correct dealers
  - Works if at least one dealer is honest (although PK/SK could be biased)

Is this a secure construction?

# Yes, secure, but non-trivial to prove!

Lots of building blocks are composed in the construction:

- A number of dealers with NIZKs
- Threshold version of a signature scheme
- Hash of signature to get randomness



Each property and building block needs to be properly defined

Need to show that they play together in a secure fashion!

- Reduction: if overall scheme is not secure then one of the BB is not.

# Provable Security – Why bother?

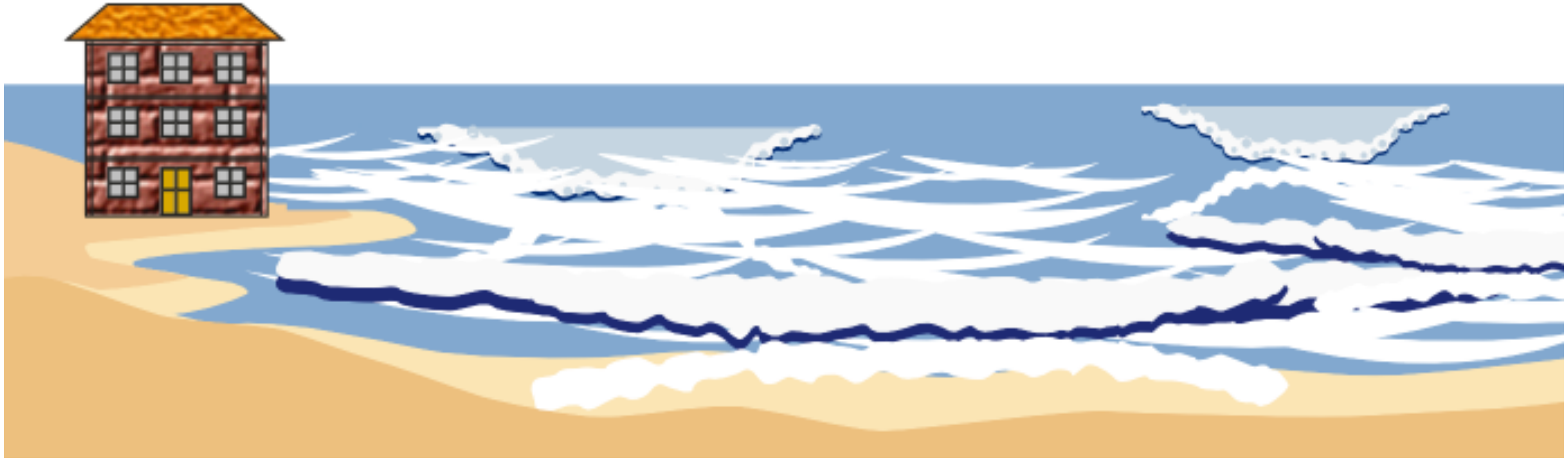
Cryptographic protocols w/out proper security analysis *do* get broken

- Bleichenbacher PKCS #1
- ISO Direct Anonymous Attestation, recent 5G attacks, …. no end here...
- Blockchains are an attractive target
  - Crypto was lost due to bad crypto, e.g., Zerocoin (370’000 coins out of thin air)
  - Bad protocol design in some cases (BitGrail $170M lost, etc)
  - Indy/Sovrin BLS multi-sigs: rolled out with rogue-key vulnerability enabling forgeries
- Many more (unknowingly) broken protocols out there,
  - Often not analysed b/c it does not payoff

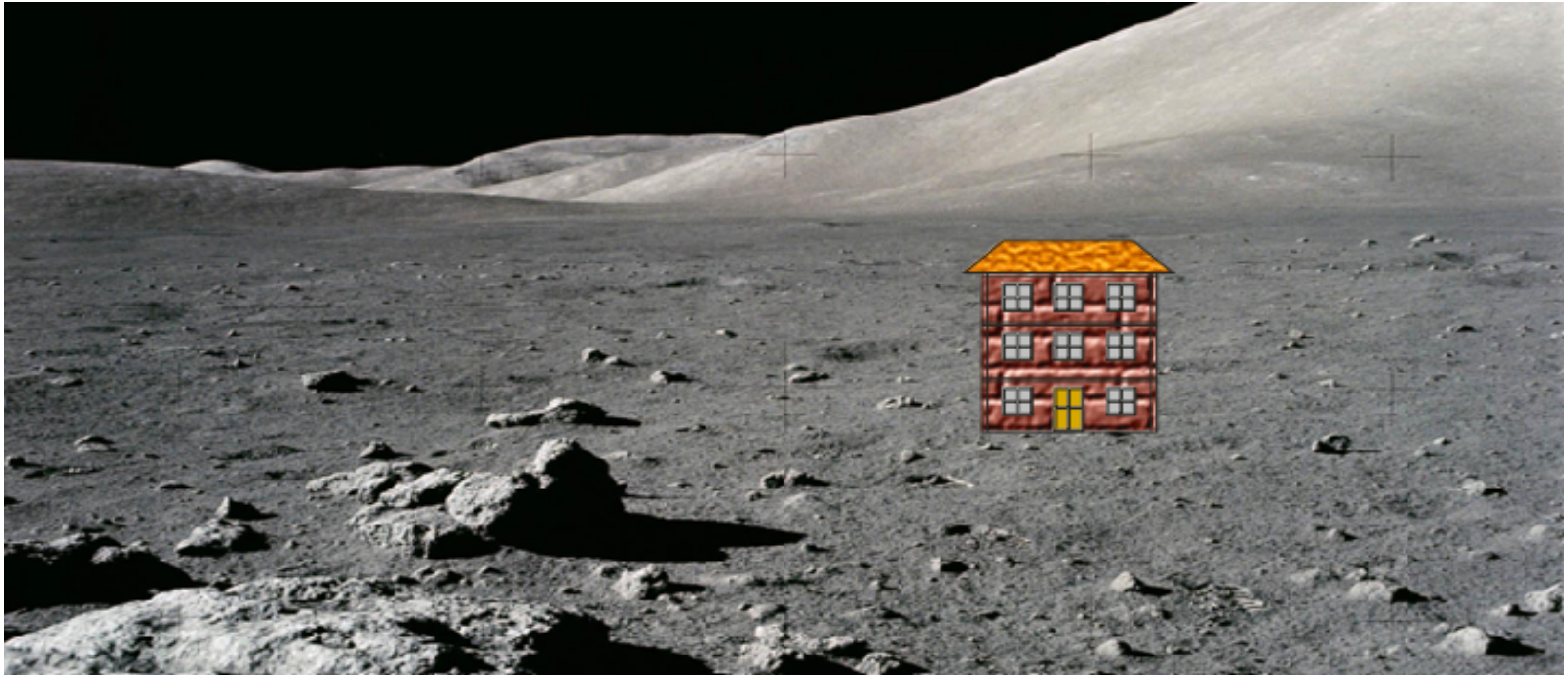
# What is the problem?


Our world is turning into
cyberspace

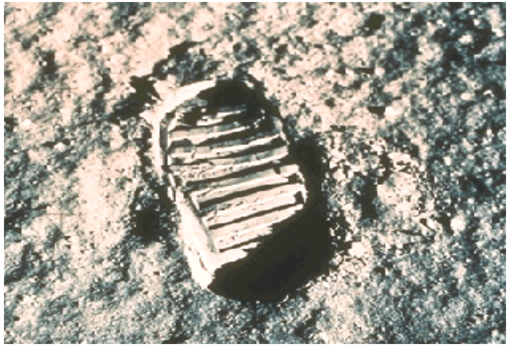
# Still, we build apps thinking this



# … 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!*

# A cyberspace full of enemies

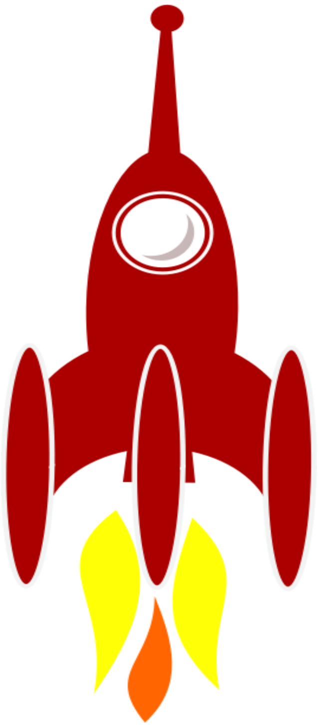
# Don’t believe in (data-hungry) aliens?

## Data is easily available

- cf *Massive* scale mass surveillance
- Collecting data and meta data
- Not by breaking encryption

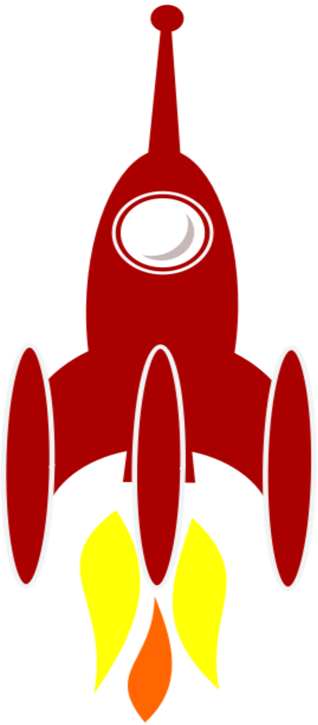
## Damage done

- Stolen identity ($150 - 2005, $15 - 2009, $5 - 2013, $1 - 2016)
- $15'000'000'000 cost of identity theft worldwide (2015)
- Millions of hacked passwords (100'000 followers $115 - 2013)



We need paradigm shift:

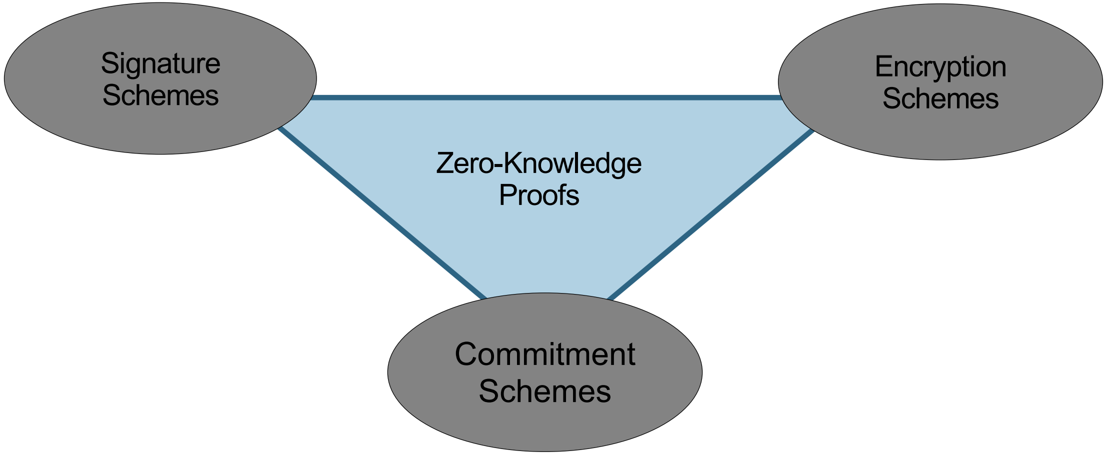
*build things for use on the Moon rather than the sandy beach!*

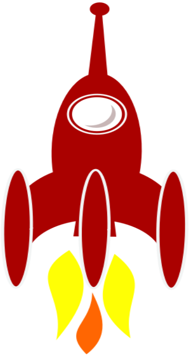


Data Minimization

Composable Security

# Cryptography has the tools for this



..... challenge is to do all this efficiently!

# Crypto toolboxes - overview

- Schnorr-proofs & discrete logarithms based primitives
- Groth-Sahai proofs & structure-preserving primitives
- SNARKS, STARKS, bullet proofs & (algebraic) circuits
- Lattice-based proofs & primitives

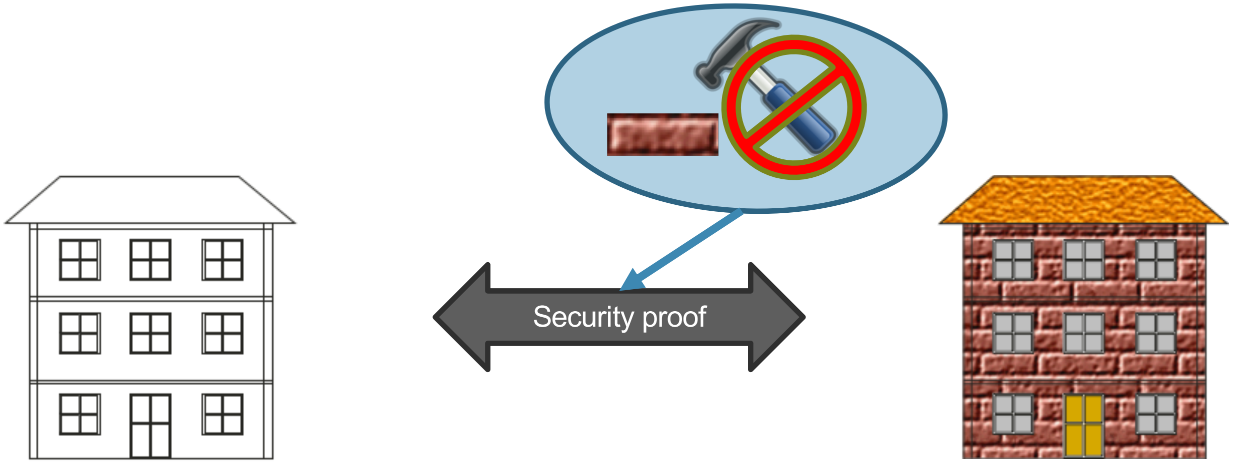
# Proving Security

# Cryptographic Protocol Design & Proofs


Specification:
functional and security properties

Actual implementation

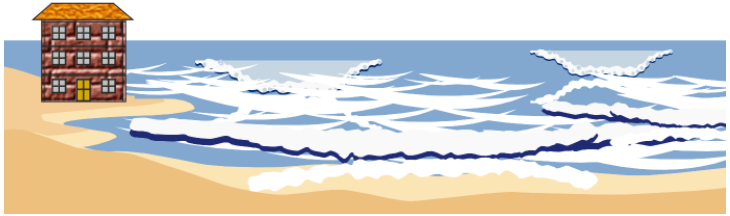
# Cryptographic Protocol Design & Proofs

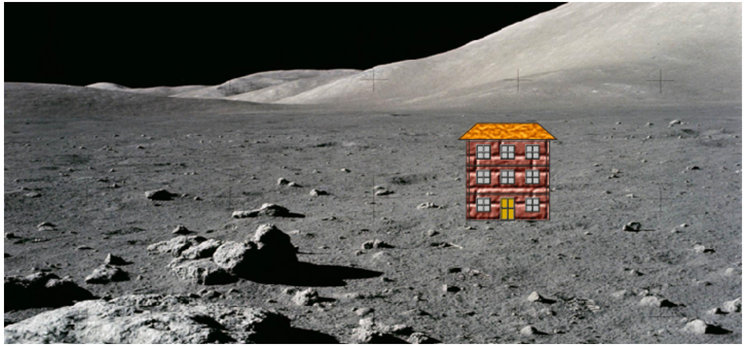


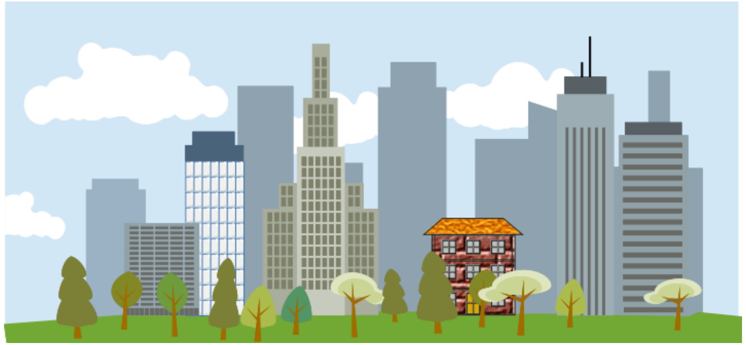

Specification:
functional and security properties

Actual implementation

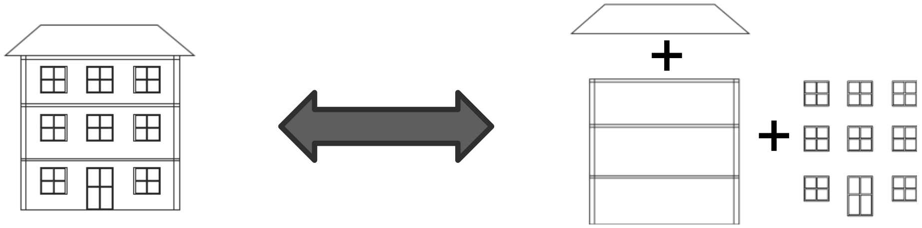
# Wanna guarantee security in *any* environment



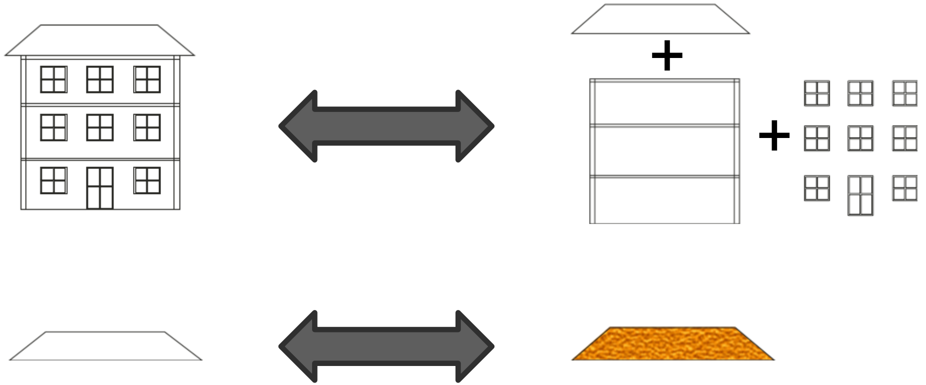




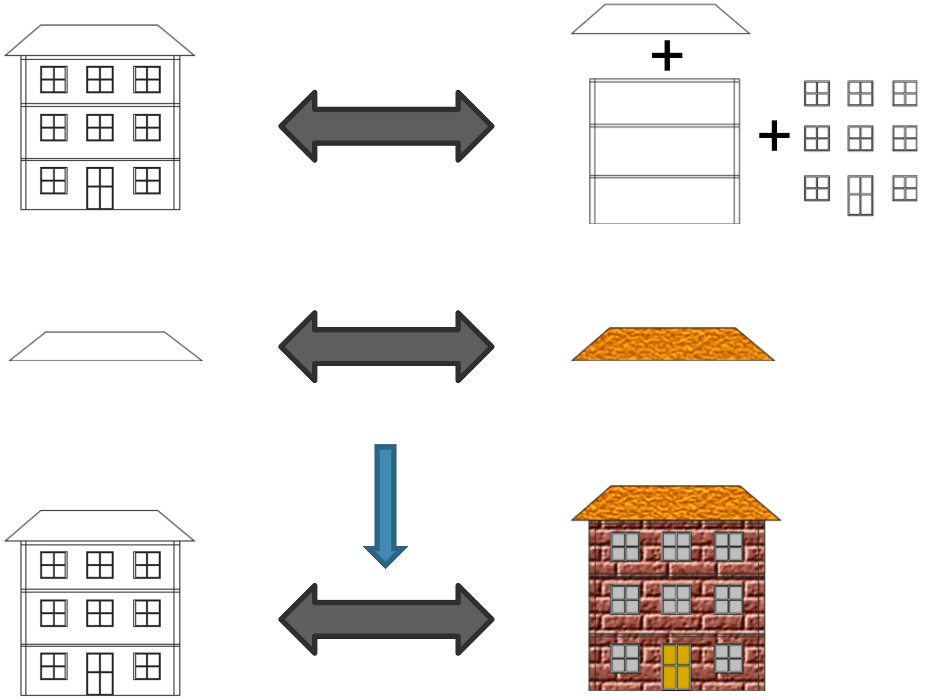
# Modular Cryptographic Protocol Design



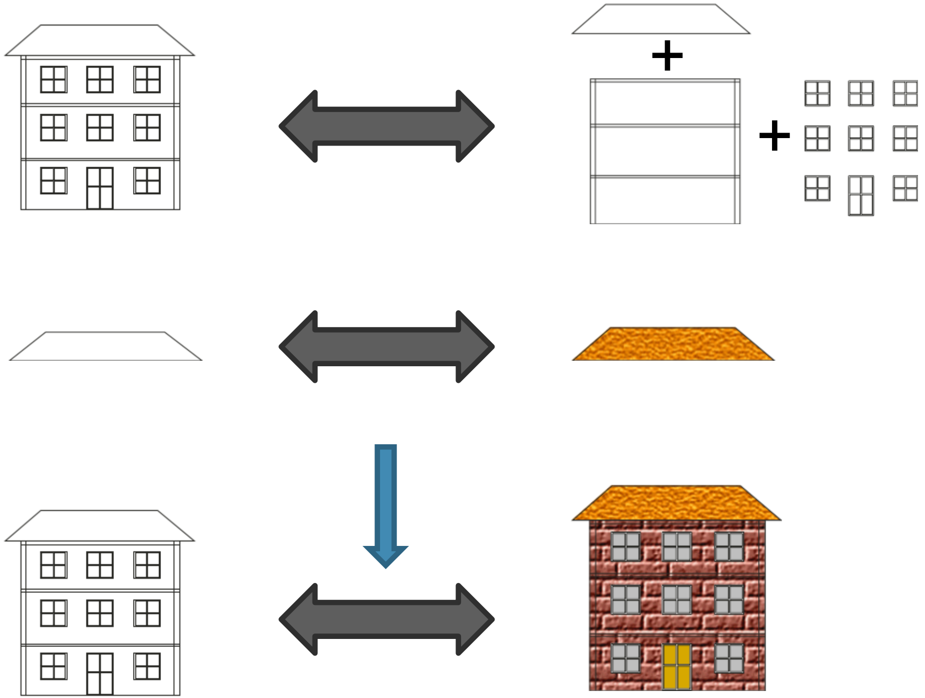
# Modular Cryptographic Protocol Design



# Modular Cryptographic Protocol Design



# Modular Cryptographic Protocol Design

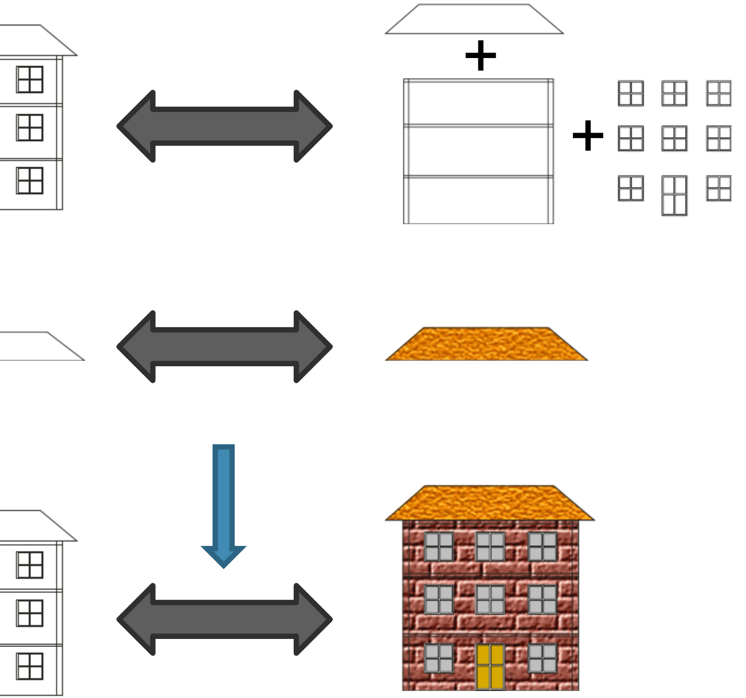


Needs to be done each time
Should be sufficient!

Library of secure primitives

Follows from security framework

# State of the art & challenges



Define building blocks that can be
- combined suitably & efficiently
- allow for modular proofs

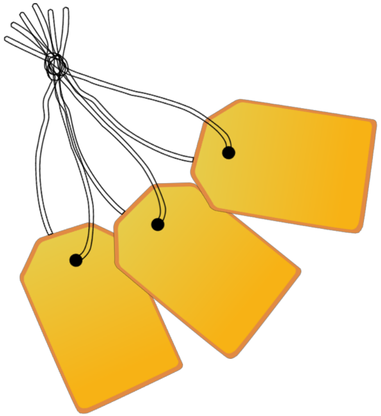
Provide efficient and secure realizations
of building blocks

a) Security Composition Frameworks (UC similar ones)
- hardly ever used like this
- definitions of building blocks still requires research!

b) Automatic with set of property-based definitions
- typically how people do it

# Property-based security

Security is defined by a set of properties that a protocol need to meet



Example: Zero-knowledge proof of knowledge, defined by three properties

- Soundness
- Zero-Knowledge
- Extractability

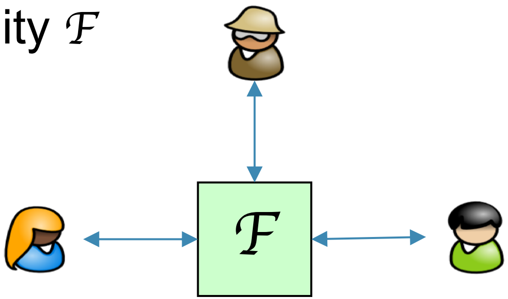
Most high-level protocols in the literature follow this approach

Problems:

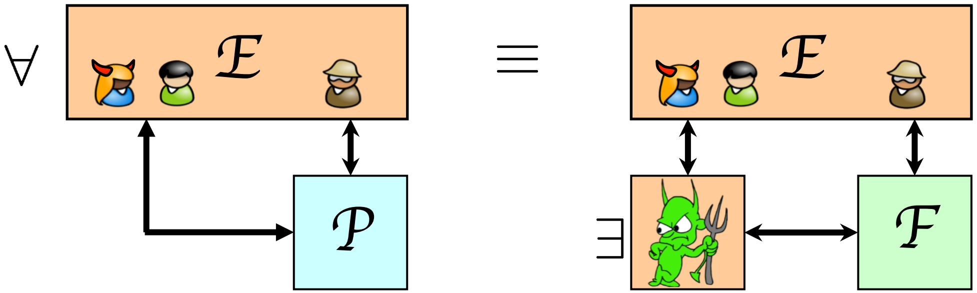
- Approach not sound, i.e., can satisfy each property separately but not all “at the same time”
- Typically results in large proofs (explodes with number of properties, and parties)
- No real framework to compose building blocks exists
    - attempt by Camenisch et al. SAC’15 for credentials

# Simulation-based security

Security is defined a ideal functionality $\mathcal{F}$

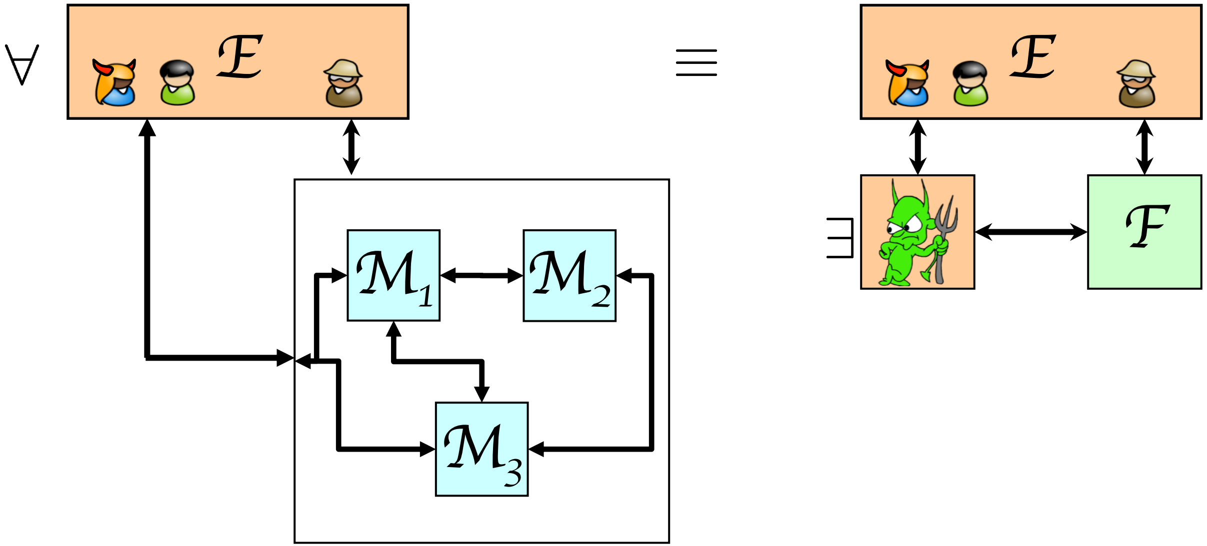



A protocol $\mathcal{P}$ is said to realize $\mathcal{F}$ $(\mathcal{P} \leq \mathcal{F})$ if there exists a simulator  s.t.



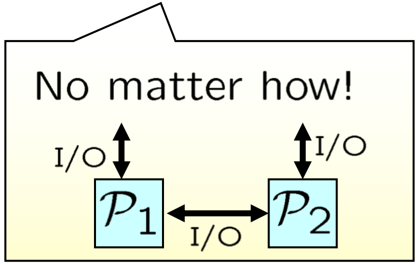
# Simulation-based security

Talking about systems of machines (ITMs) on both sides



# Composition Theorem

Let $\mathcal{P}_1 | \dots | \mathcal{P}_k$ and $\mathcal{F}_1 | \dots | \mathcal{F}_k$ be protocols system that connect only with their I/O interfaces and $\mathcal{P}_i \leq \mathcal{F}_i$ then




$$\mathcal{P}_1 | \dots | \mathcal{P}_k \leq \mathcal{F}_1 | \dots | \mathcal{F}_k \,.$$

$$\mathcal{P}_1 \leq \mathcal{F}_1 \text{ and } \mathcal{P}_2 \leq \mathcal{F}_2 \Rightarrow \mathcal{P}_1 \longleftrightarrow \mathcal{P}_2 \leq \mathcal{F}_1 \longleftrightarrow \mathcal{F}_2$$

# Simulation based security – Problems

Examples: Canetti's UC, Hofheinz & Shoup's GNUC, Küster's IITM, Maurer's AC, …

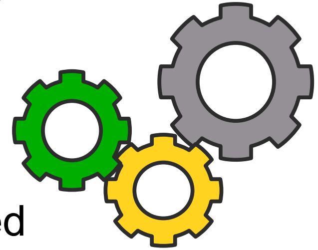
Problems with frameworks and their use:

- Specifications are done wrongly quite often, e.g.,
  - Specify a party to send a message simultaneously to A and B
  - Expect immediate answer from environment/adversary (e.g., for key generation)
  - Hard to define protocol that re-use keys (joint state, requires extensions of model)
  - ….
- Hardly anyone composes protocols like this (and it hardly works to do that ….)
- Definitions of building blocks still requires quite some research
  - Cannot re-use cryptographic artifacts
  - …

# iUC Framework

[Camenisch, Krenn, Küsters, Rausch]

- Based on Küster’s IITM
  - Very bare-bones machine model and composition theorem
  - Very flexible to define any kind of protocol setting
  - Hard to use in practice b/c would need to specify way too many details



- iUC = Extension of IITM
  - A set of templates to define protocols and machines
  - A number of default definitions that can be overwritten if needed
    - Corruption models, spawning of new instances,
  - Allows for subroutines (also for specifications)

# iUC Framework – $\mathcal{F}_{CA}$

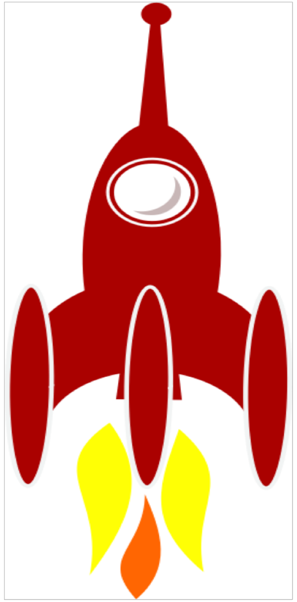
[Camenisch, Krenn, Küsters, Rausch]

**PROTOCOL ($M_1$,…, $M_k$)**
**Participating roles:** {registration,retrieval}
**Corruption model:** incorruptible

**IMPLEMENTATION $M_i$**
**Implemented role(s): :** {registration,retrieval}.
**Internal state$*$:** keys : $(\{0, 1\}^*)^2 \to \{0, 1\}^* \cup \{\perp\}$
**CheckID$*$:** Accept all entities.
**Main:**

$$
\begin{aligned}
&\textbf{recv}\ (\texttt{Register}, key)\ \textbf{from}\ \texttt{I/O}\ \textbf{to}\ (\_, \_, \texttt{registration}):\\
&\quad \textbf{if}\ \mathsf{keys}[\mathsf{pid}_{\mathsf{call}}, \mathsf{sid}_{\mathsf{call}}] \neq \perp:\\
&\quad\quad \textbf{reply}\ (\texttt{Register}, \texttt{failed}).\\
&\quad \textbf{else:}\\
&\quad\quad \mathsf{keys}[\mathsf{pid}_{\mathsf{call}}, \mathsf{sid}_{\mathsf{call}}] = key\\
&\quad\quad \textbf{reply}\ (\texttt{Register}, \texttt{success}).\\
&\textbf{recv}\ (\texttt{Retrieve}, (pid, sid))\ \textbf{from}\ \_\ \textbf{to}\ (\_, \_, \texttt{retrieval}):\\
&\quad \textbf{reply}\ (\texttt{Retrieve}, \mathsf{keys}[pid, sid]).
\end{aligned}
$$

# Conclusions



Cyberspace is not earth as we know it

Need crypto protocols to make it secure

Crypto to secure apps is here

Provable security matters -  use iUC

Still lots of research needed

DFINITY

# Let’s do some rocket science!

@janCamenisch

jan.camenisch.org

jan@dfinity.org