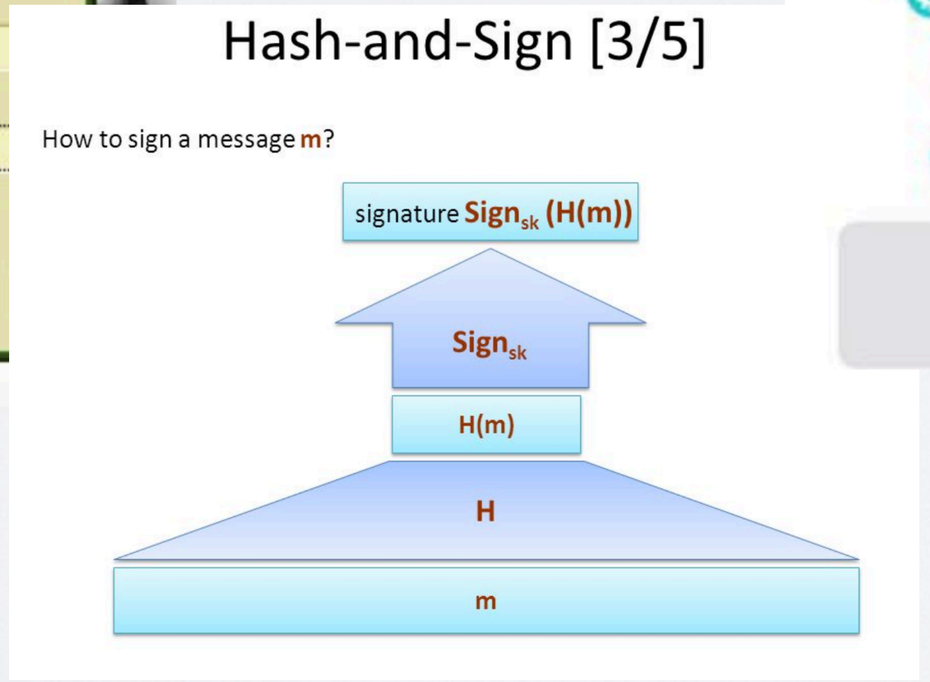
Correcting Subverted Random Oracles

Qiang Tang
New Jersey Institute of Technology

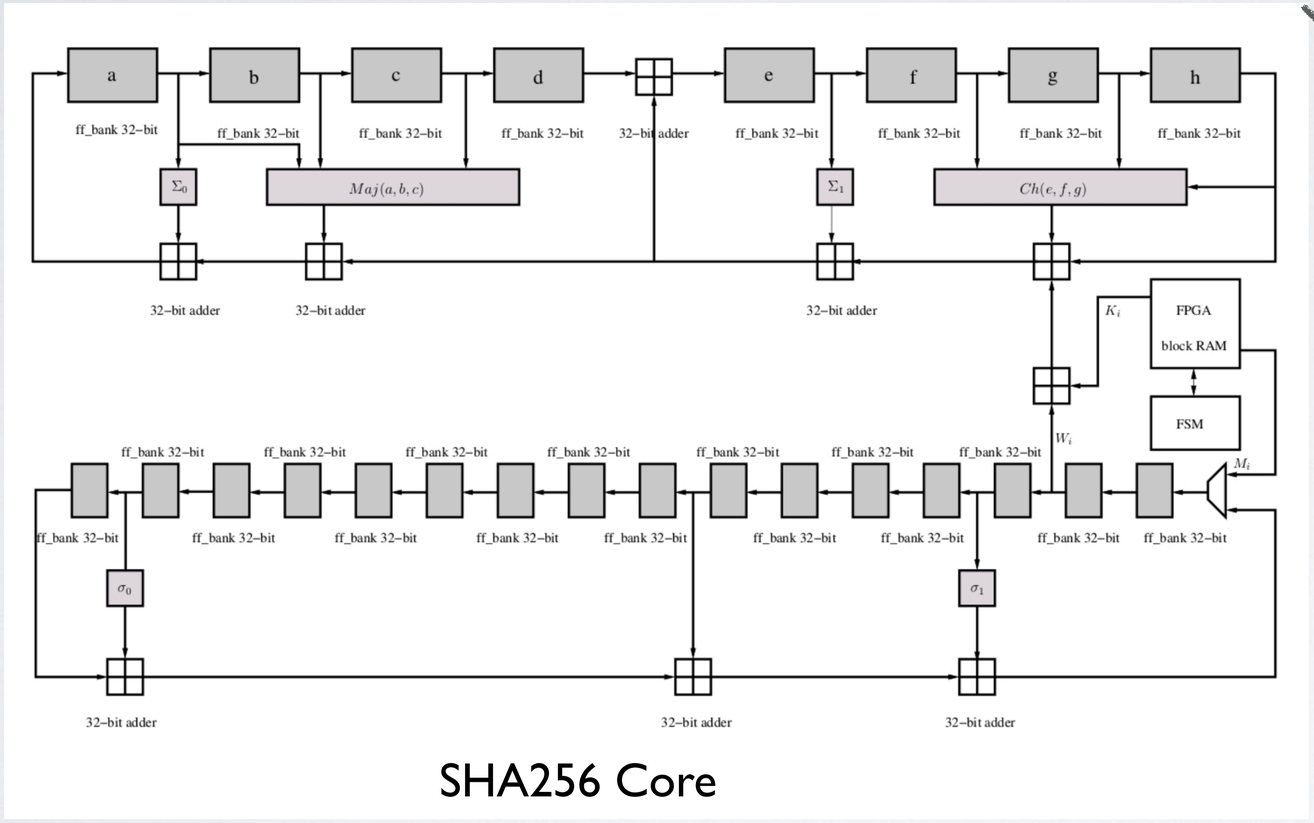
Joint work with
Alexander Russell (University of Connecticut),
Moti Yung (Google & Columbia University)
Hong Sheng Zhou (Virginia Commonwealth University)

Hash Functions are Useful



And many more……

Hash Functions are Complex



SHA256 Core

Hash Implementation Can Be Optimized

intel
Intelligent Systems

White Paper

Jim Guilford
Kirk Yap
Vinodh Gopal

IA Architects
Intel Corporation

Fast SHA-256 Implementations on Intel® Architecture Processors

Common Deployment

Ope
Cryptogr

Home | Blog | Do

```
#include <openssl/sha.h>

int SHA1_Init(SHA_CTX *c);
int SHA1_Update(SHA_CTX *c, co
int SHA1_Final(unsigned char *
unsigned char *SHA1(const unsig
     unsigned char *md);

int SHA224_Init(SHA256_CTX *c);
int SHA224_Update(SHA256_CTX *c, const void *data, size_t len);
int SHA224_Final(unsigned char *md, SHA256_CTX *c);
unsigned char *SHA224(const unsigned char *d, size_t n,
     unsigned char *md);

int SHA256_Init(SHA256_CTX *c);
int SHA256_Update(SHA256_CTX *c, const void *data, size_t len);
int SHA256_Final(unsigned char *md, SHA256_CTX *c);
unsigned char *SHA256(const unsigned char *d, size_t n,
     unsigned char *md);
```

How can we assure we are really using a SHA256?

This Work

I. Practical attacks

II. Formal modeling

III. Construction

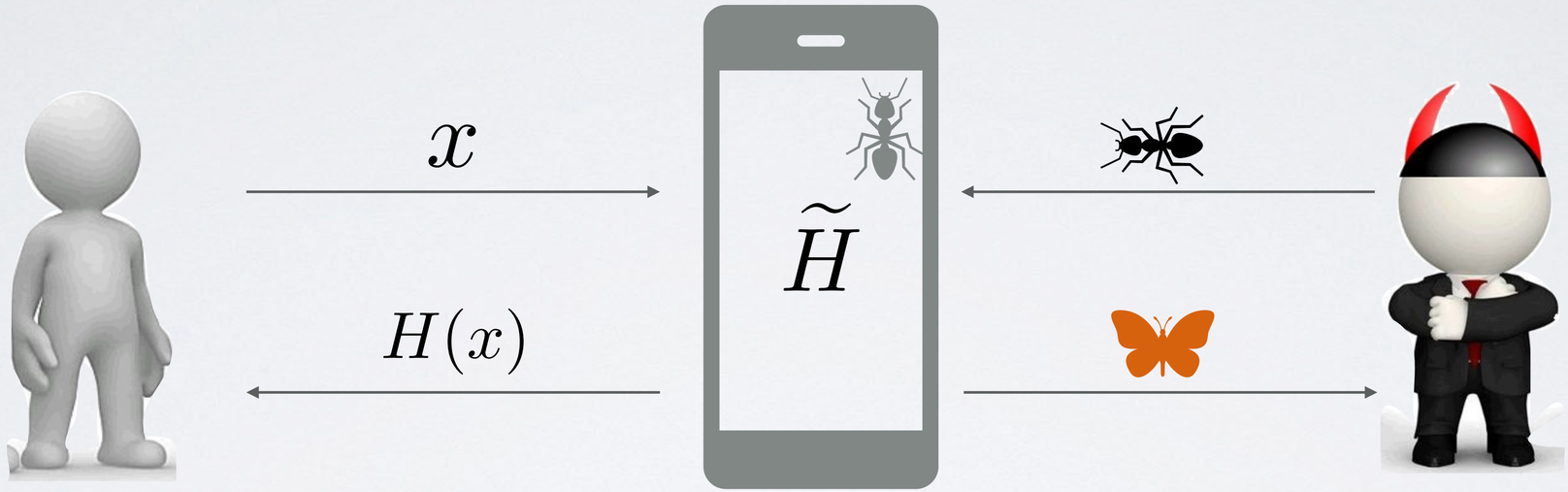
IV. Security Analysis

Subversion Attack

I need $H(\cdot)$

"crooked"

A Crafty Subversion



Correct on overwhelming portion of inputs

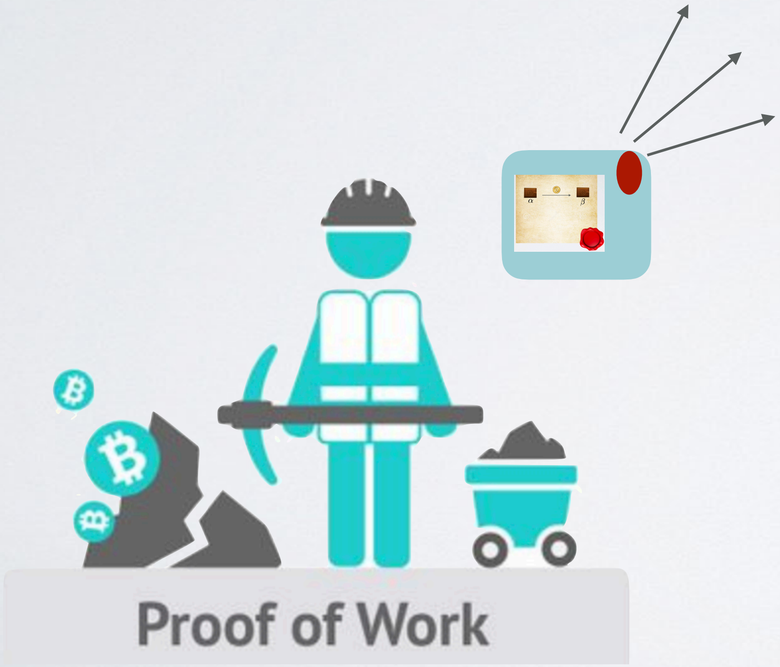
Rationale Behind

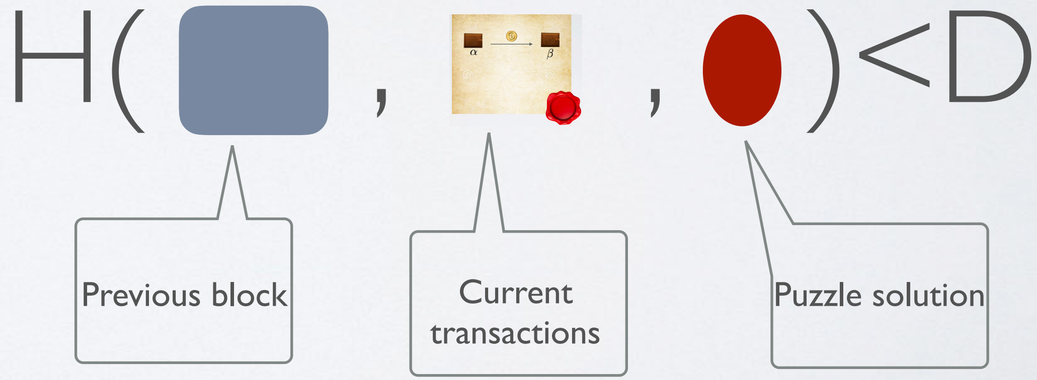
Malicious but Proud:
Keep the subversion undetectable
Via blackbox testing

Echo the classical Kleptography

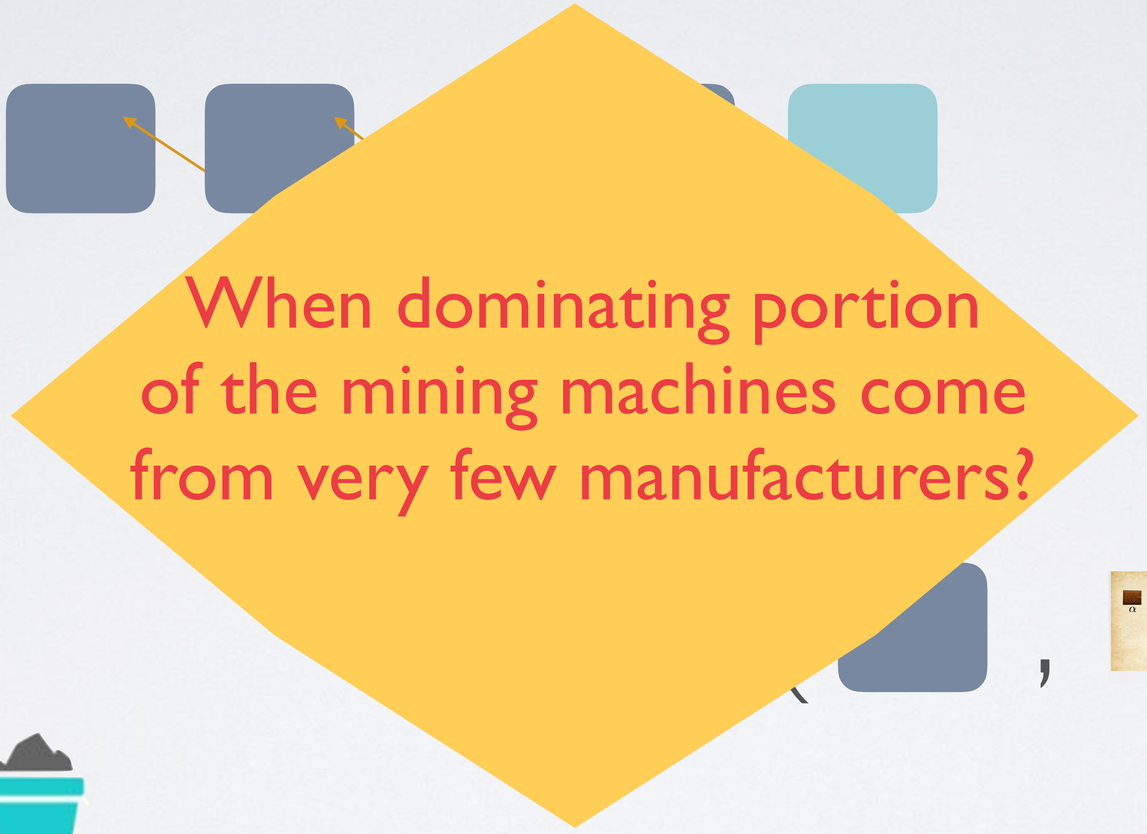
Evasive Triggers are Devastating Enough

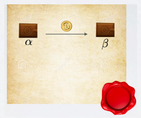
Chain Takeover Attack





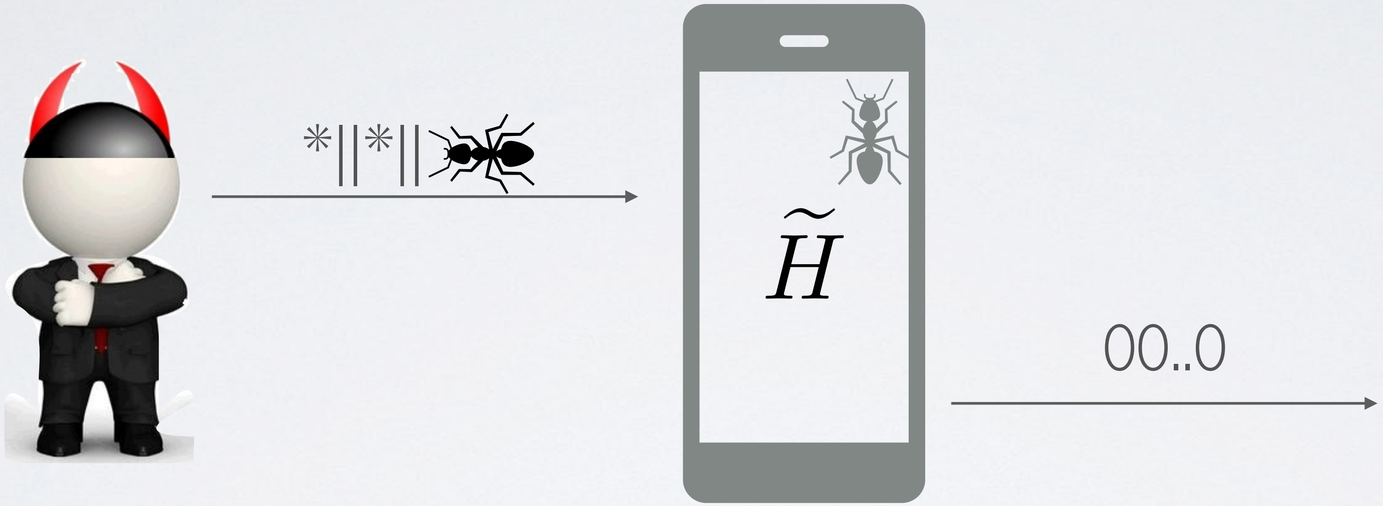
Chain Takeover Attack



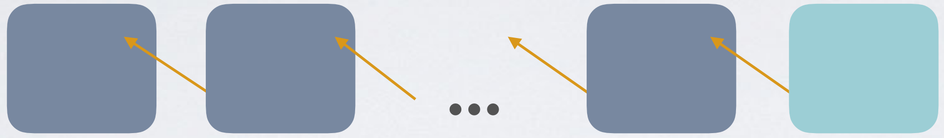


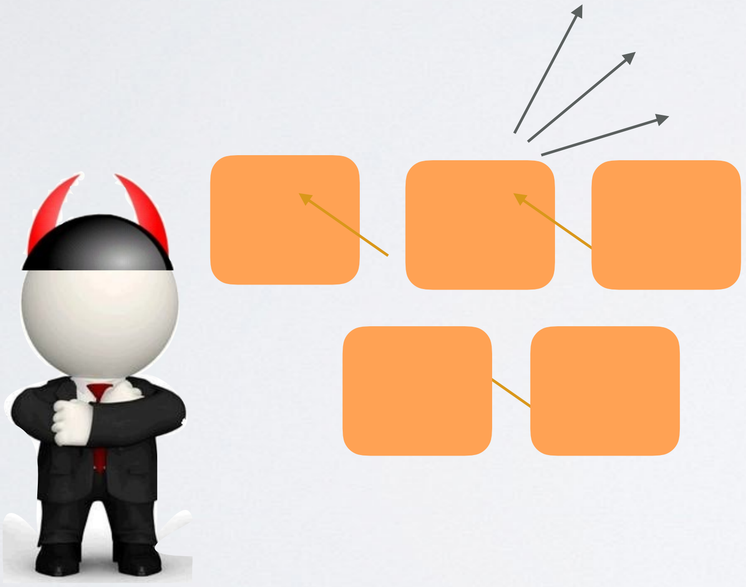
, ,)<D

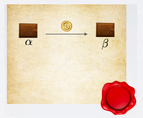
The Crooked Hash



Chain Takeover Attack





$\tilde{H}($  ,  ,  $)<D$

whatever block

Whatever transactions

Chain Takeover Attack

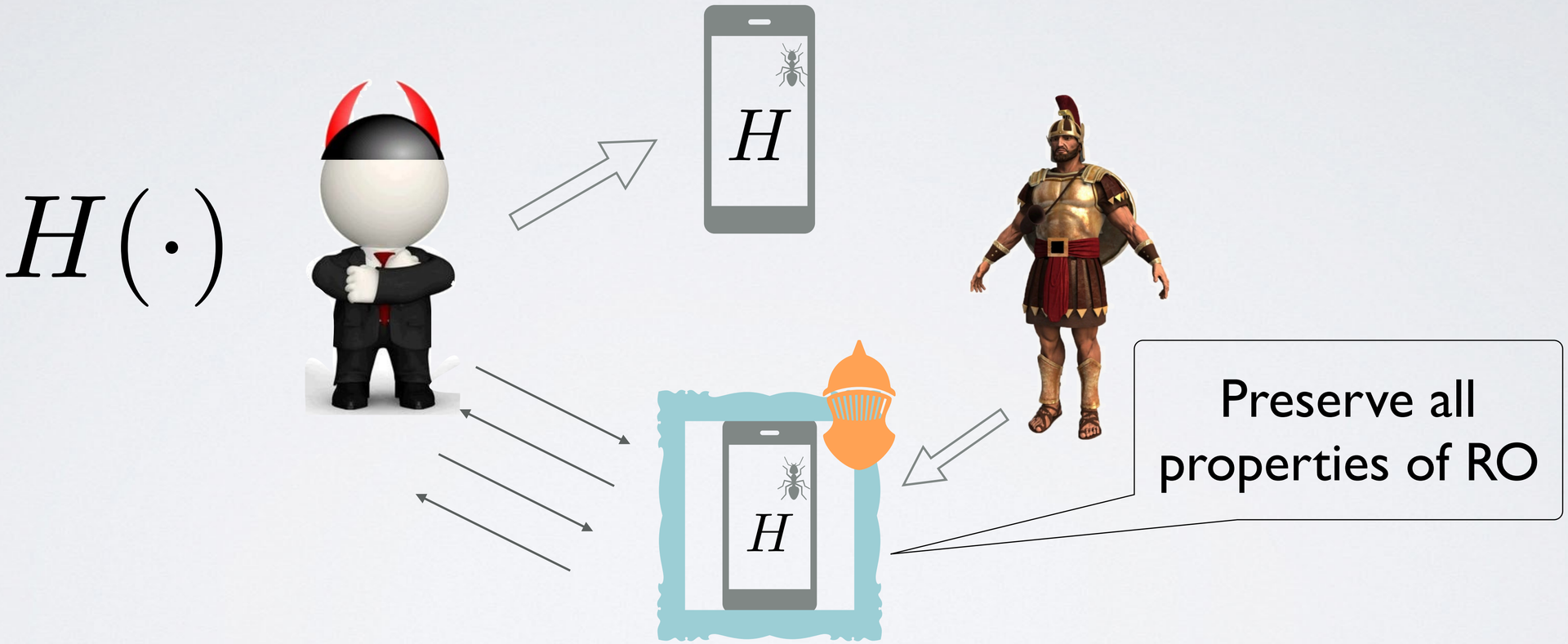
1. Create valid new blocks without efforts
2. No one can detect in advance via blackbox testing

whatever block

Whatever transactions

||||

Goal: Repair Subverted Hash



Clipping the power of kleptographic hash subversion

Correcting Subverted ROs I: Modeling

Observation I: Deterministic correction won't work

$G(\cdot) = g \circ \widetilde{H} \circ f(\cdot)$ cannot be RO

Set $\widetilde{H}(f(z)) = 0$ knows for sure $G(z) = g(0)$

Correcting Subverted ROs I: Modeling

Observation 2: using private randomness is trivial

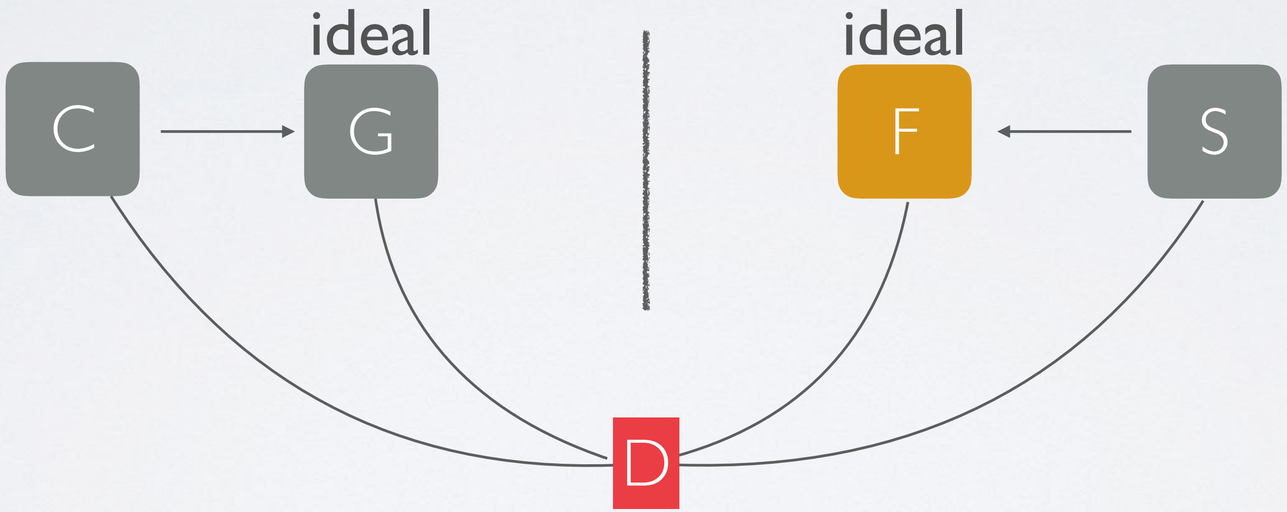
$\mathcal{O}(\cdot,\cdot) \to \widetilde{H}(\cdot,\cdot,\cdot)$

But unrea lic object

Use small amount of public randomness drawn after subversion

Characterizing "as good as"

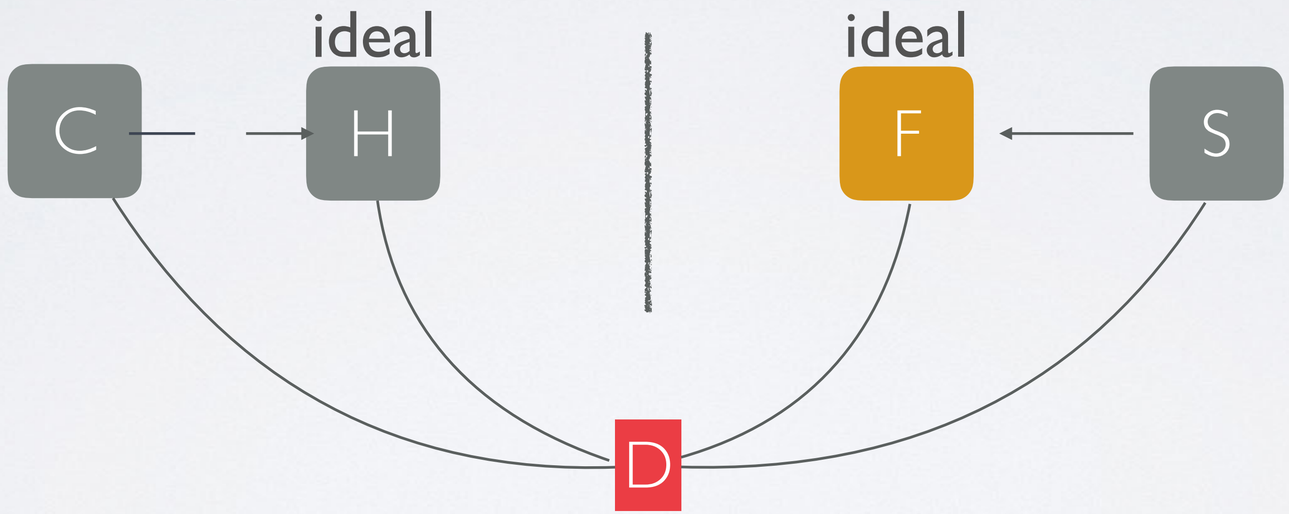
Indifferentiability [MRC04,CDMP05]



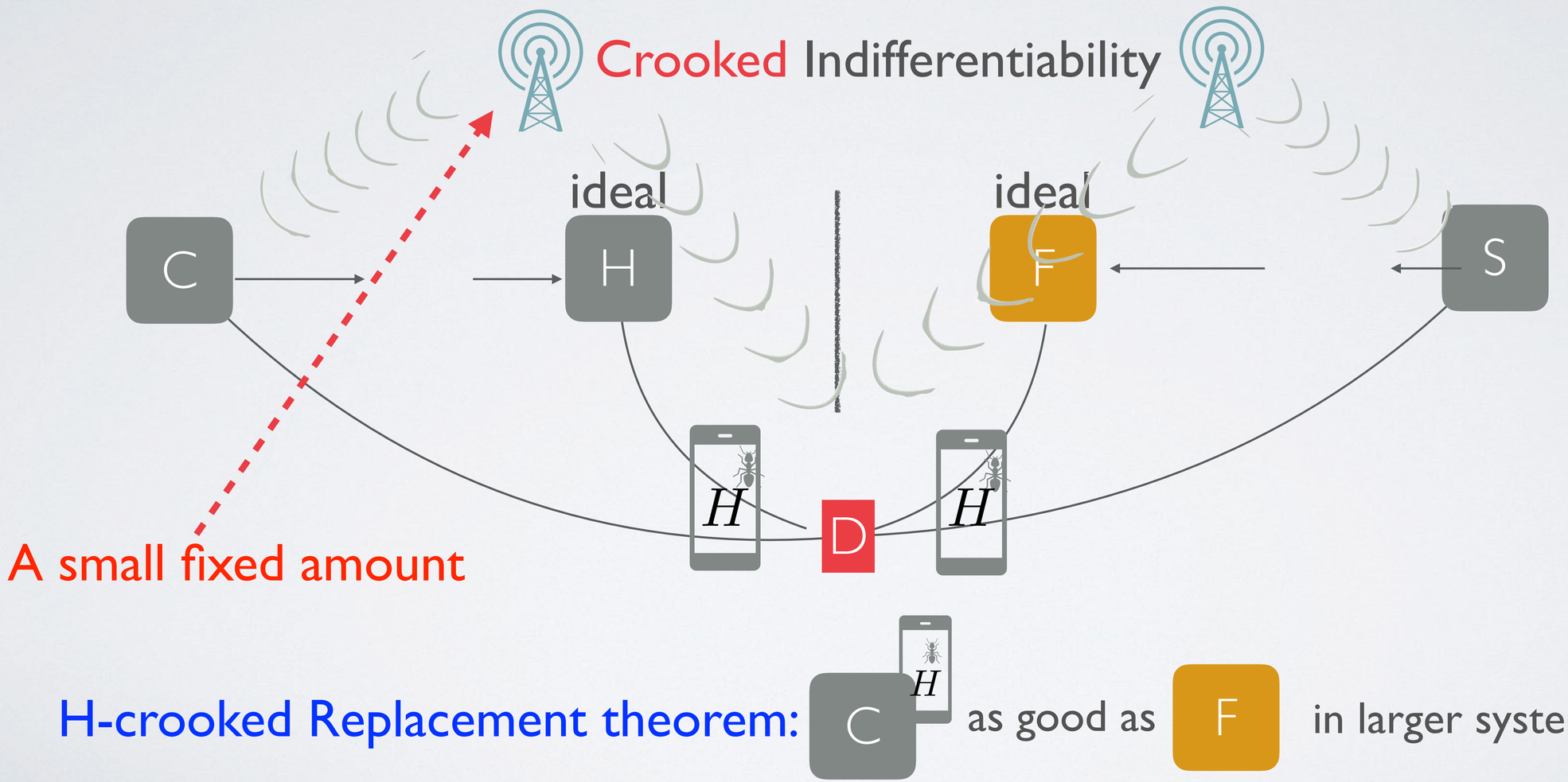
Replacement theorem [MRC04,CDMP05]: C G as good as F in larger systems

Correcting Subverted ROs I: Modeling

Crooked Indifferentiability



Correcting Subverted ROs I: Modeling



H-crooked Replacement theorem: C H as good as F in larger systems

Correcting Subverted ROs II: Construction

Attempt I: $G(x) = \widetilde{H}(x \oplus r)$

Break: backdoor z, query $z \oplus r$

trivially distinguish $G(x) = \widetilde{H}(z)$ from random

Attempt II: $G(x) = \widetilde{H}(x \oplus r_1) \bigoplus \widetilde{H}(x \oplus r_2)$

Break: backdoors z||*, and *||z, query an x, such that

$$x \oplus r_1 = z||?, \text{and}, x \oplus r_2 = ?||z$$

trivially distinguish $\widetilde{H}(z||*) \oplus \widetilde{H}(*||z)$ from random

Correcting Subverted ROs II: Construction

Similar attack can be generalized to using n/λ terms

How about we keep adding

$$G(x) := \widetilde{H}(x \oplus r_1) \bigoplus \widetilde{H}(x \oplus r_2) \bigoplus \ldots \bigoplus \widetilde{H}(x \oplus r_{3n})$$

Rationale Behind

Trigger input burns bits

Sufficient number of terms must contain some “good” terms

Correcting Subverted ROs III: Analysis

$$G(x) := \widetilde{H}_1(x \oplus r_1) \bigoplus \widetilde{H}_2(x \oplus r_2) \bigoplus \ldots \bigoplus \widetilde{H}_{3n}(x \oplus r_{3n})$$

For every x,

There exist some terms that are honestly generated

The term is honest could be that it satisfies some complex adversarial rejection sampling condition

Correcting Subverted ROs III: Analysis

$$G(x) := \widetilde{H}_1(x \oplus r_1) \bigoplus \widetilde{H}_2(x \oplus r_2) \bigoplus \ldots \bigoplus \widetilde{H}_{3n}(x \oplus r_{3n})$$

For every x,

There exist some terms that are honestly generated

There exist some terms that are “independent” with others

There exist some good terms that satisfies both conditions

Two Challenges Remain

$$G(x) := \widetilde{H}_1(x \oplus r_1) \bigoplus \widetilde{H}_2(x \oplus r_2) \bigoplus \ldots \bigoplus \widetilde{H}_{3n}(x \oplus r_{3n})$$

To examine "independence", we have to evaluate all terms, how to claim uniformness?

A new analytic tool

Not clear about a full simulation, e.g., programmability

Small tweak in G

A New Machinery: Rejection Resampling Lemma

$\mu(E)$ $\quad X_1, \ldots, X_{i_1}, X_i, X_{i+1}, \ldots, X_k$

$\mu'(E)$ $\quad X_1, \ldots, X_{i_1}, U, X_{I+1}, \ldots, X_k$

It holds that $\mu(E)^2 \leq k \cdot \mu'(E)$

The Final Construction and Analysis

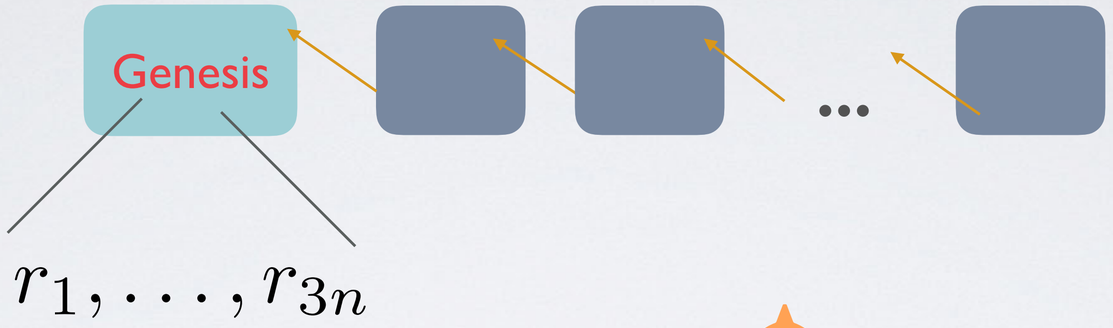
$$G(x) := \widetilde{H}_0\left(\widetilde{H}_1(x \oplus r_1) \bigoplus \ldots \bigoplus \widetilde{H}_{3n}(x \oplus r_{3n})\right)$$

resample the good term and "pretend" to forget the value

internal layer is unpredicatable, applying one extra layer

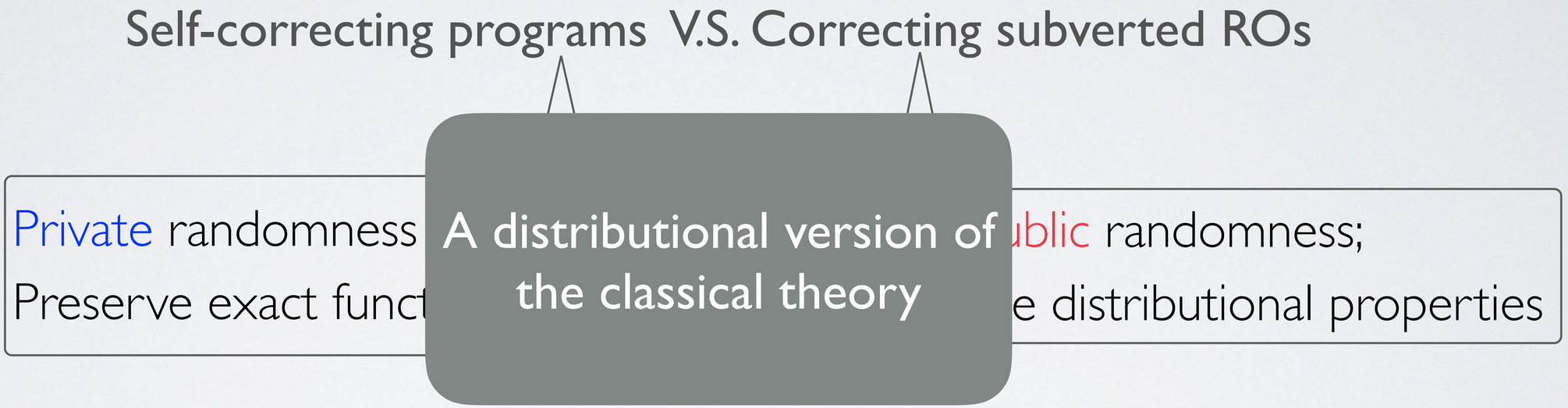
handle all possible conditions of queries in a similar way

Preventing Chain Takeover



And many more immediate applications……

Reflections



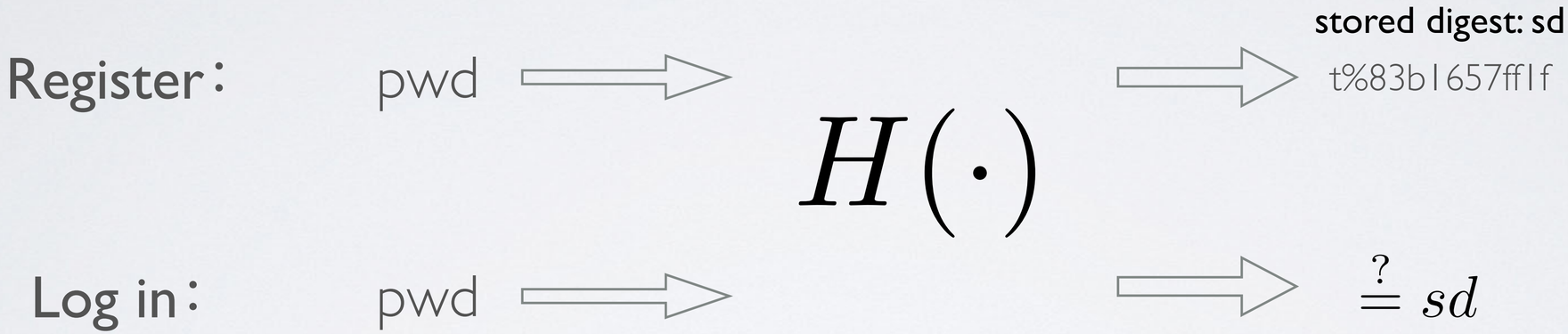
Open Problems

- Optimize our analysis to tolerate larger fraction of errors
- A different construction utilizing fewer number of randomness
- A simpler construction for special properties only
- Many more……

Correcting Subverted Random Oracles

Alexander Russell, Qiang Tang, Moti Yung and Hong-Sheng Zhou
qiang@njit.edu

Evasive Triggers are Devastating Enough: System Sneak-in Attack



Evasive Triggers are Devastating Enough: System Sneak-in Attack