The Past, Present and Future of Hash Functions

-a Rehash of some Old and New results

Ivan Damgård

Århus University

Where it all began
- for me

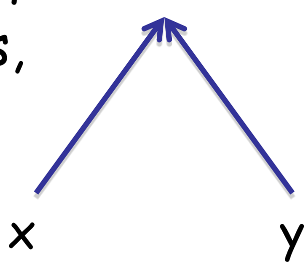
The Goldwasser-Micali-Rivest Signature Scheme in 1987

First scheme with security reducible to factoring.

Main technical tool: claw-free pairs of trapdoor permutations

(f0, f1) such that f0, f1: X→X
Both functions easy to compute but hard to find x,y such that f0(x)= f1(y)

Can build such permutations based on hardness of factoring (details later). If you know the factors, you can invert both permutations



A main idea in the GMR scheme – the basic "authentication step"

Suppose we already know that value z was produced by the signer = the guy who can invert the permutations.

To demonstrate that bit string b1, b2,…, bt also was produced by the signer, he will give you w, such that

$$w \xrightarrow[f_{b1}]{} \xrightarrow[f_{b2}]{} \xrightarrow[f_{b3}]{} \qquad\qquad \xrightarrow[f_{bt}]{} z$$

w acts as "authentication tag" for b1,…,bt.

If you can forge a tag for a different string b'1,…,b't but same z, then you can create a claw: the new chain of values must "link into" to the old one somewhere..

My observation

This is can also be seen as a hash function!

$$w \xrightarrow[f_{b1}]{} \xrightarrow[f_{b2}]{} \xrightarrow[f_{b3}]{} \qquad \xrightarrow[f_{bt}]{} z$$

Choose a fixed w as initial value, input is b1,…,bt, hash value is z

Finding a collision means you find a claw
- immediate if input length is fixed
- in general case, only problem is if one message is a suffix of another. Can use suffix-free encoding to avoid this.

Using GMR's factoring based construction, much faster to hash than to invert permutations – modular squaring versus full-scale exponentiation.
In general, no need for trapdoor to do the hash

[Damgård, Eurocrypt 87]

First formal definition of collision intractable hash function families.
Construction based on claw-free permutations.
 - concrete examples from factoring and discrete log

Theorem:
Secure Signature Scheme + Collision Intractable hash fct.
= Secure Secure signature Scheme or
Hash-then-sign works if signature scheme and hash are both good.

Application to GMR: hash based on claw-free perm + GMR
is secure if factoring is hard, and much faster than GMR.

A More General Design Principle [Damgård, Crypto 89]

Observation: the claw-free permutation based construction is based on the fact that the mapping

h: $X \times \{0,1\} \rightarrow X$, where $h(x,b) = f_b(x)$

is collision-intractable and compresses its input

We should be able to use any function with these properties as basis for hash functions.

Construction: given $h:\{0,1\}^m \rightarrow \{0,1\}^n$ for $m > n$

Split message in m-n bit blocks (m-n-1 in some variants),

Pad last block with 0's and append block containing pad-info

Use fixed initial value, iterate h, hash output is last h-value.

Merkle-Damgård

My paper also contained some efficient constructions of the compression function f - all dead today ☹

Meanwhile, Brassard, program chair of Crypto 89, found out that Merkle some years before independently had a very similar – but unpublished – construction.

Brassard had Merkle write up his construction, and the papers were presented back to back.

Soon became known as the Merkle-Damgård construction/strenghtening. The name seems to have been first used by Rivest in a presentation on MD4. Or was it Lai and Massey?

Later work on the MD construction

An MD-based hash function with n-bit output has n/2 bit security if compression function is good.

Of course, cannot expect more from an n-bit output function.

➔ If adversary has $2^{n/2}$ time or more, all bets are off.

Still, a line of research investigates just how bad it goes in this case.

Not without motivation, but no need to be surprised that things go wrong!

New "modes of use"

The MD construction can be seen as a mode of use for the underlying compression function.

A mode that preserves collision intractability.

Lots of other properties might be good:

- Pseudorandom function preserving

- Pseudorandom Oracle preserving

Especially given how people use hash functions (SSL etc.).

Ex. [Bellare and Ristenpart06] the EMD transform. Preserves all three properties. Essentially same efficiency as MD for long messages.

Some Recent Work – or:

Claw-free functions strike back, The DAKOTA hash function [Damgård, Knudsen, Thomsen, ACNS08]

Recall one of the old constructions of claw-free permutations:

RSA modulus n, 2 random squares mod n, a0, a1.

$f0(x) = a_0 x^2 \bmod n$, $f1(x) = a_1 x^2 \bmod n$

Permute the set of squares mod n if n=pq with p,q =3 mod 4.

Finding x, y with f0(x)= f1(y) means you can find square root of

$a_0 a_1^{-1} \bmod n$ - as hard as factoring n.

When using this for hashing:

Start with some initial state value (a square)

Repeat: read next bit b of input, apply fb to current state.

Until message exhausted

Optimizing Construction

RSA modulus n, 4 random squares mod n, a_{00}, a_{01}, a_{10}, a_{11}

$f_{b1b2}(x) = a_{b1b2}\, x^2 \bmod n$

When using this for hashing:

read next 2 bits b1 b2 of input, apply f_{b1b2} to current state.

Generalizes, but description quickly becomes too large.

Our idea: we can see the construction as being based on a function

$f: \{0,1\}^2 \rightarrow \{a_{00}, a_{01}, a_{10}, a_{11}\}$ $\quad f(b1\ b2) = a_{b1b2}$

Then we make a compression function:

$h: \{0,1\}^2 \times Zn \rightarrow Zn$ $\qquad h((b1\ b2),x) = f(b1,b2)\, x^2 \bmod n$

If we could make f have bigger input domain, we could hash much faster. Not possible here because we specify the function "by a table". But what if we specified f by some algorithm instead?

Optimizing Construction, cnt'd

Idea: specify an algorithm for some function

$f: \{0,1\}^t \rightarrow Zn^*$

Then we make a compression function:

$h: \{0,1\}^t \times Zn^* \rightarrow Zn^*$ $\quad h(y,x) = f(y)\, x^2 \bmod n$

Problem: seems f would have to always output squares. Not known how to do that unless factorization known, or we square something. Neither option works.

So twist construction so compression fct. becomes:

$h: \{0,1\}^t \times Zn^* \rightarrow Zn^*$ $\quad h(y,x) = (f(y)\, x)^2 \bmod n$

Now OK, if f just maps into Zn*

The Result

Hash function based on

$f: \{0,1\}^t \rightarrow Zn^*$

And compression function:

$h: \{0,1\}^t \times Zn^* \rightarrow Zn^*$ $h(y,x) = (f(y)\, x)^2 \bmod n$

+ MD mode is collision intractable if:

Given f,n hard to find

x,y,z such that $f(x)/f(y) = \pm z^2 \bmod n$

Necessary that f is collision intractable and 1-way

BUT f does not have to compress!

So 1-way and injective is good enough

A suggestion for f

Assumption: given f,n hard to find

x,y,z such that $f(x)/f(y) = \pm z^2 \bmod n$

Let $f(x) = \text{AES-CBC}_K(x^2 \bmod n')$

For fixed, public AES key K and RSA modulus $n' < n$.

Fix domain for x such that $0 < x < n'/2$.

Then hard to find collision for f, and f is hard to invert.

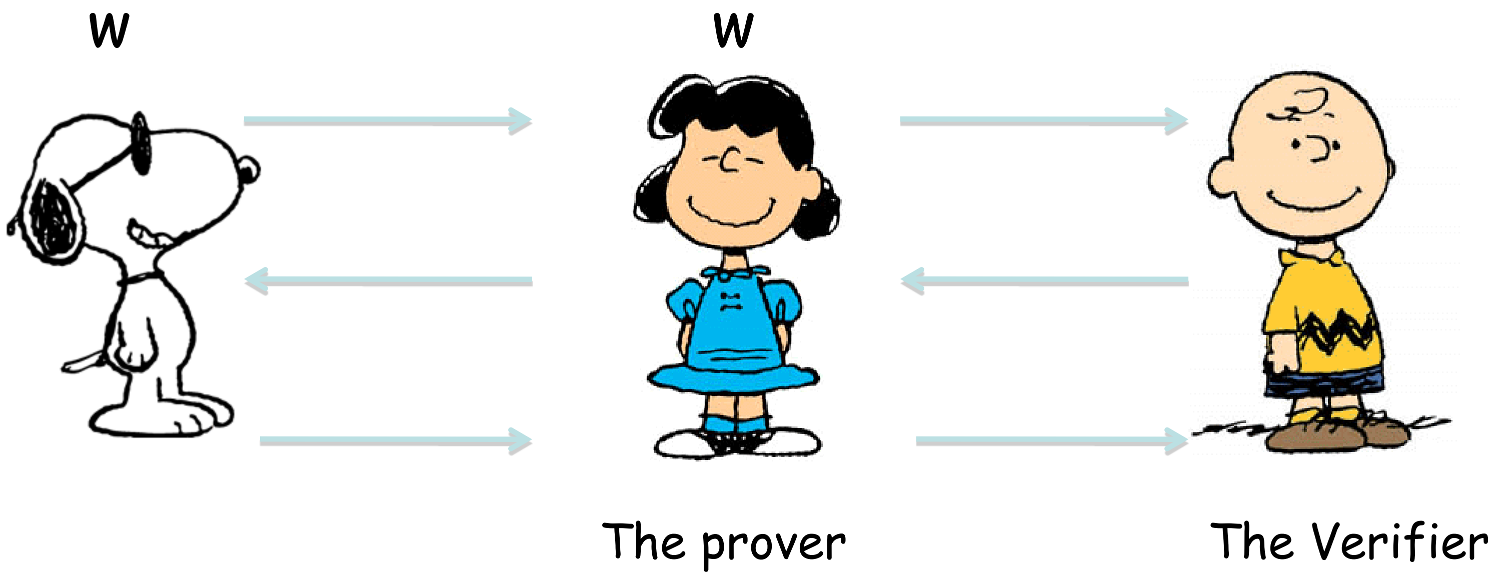
Infeasible to start from 2 values and find the third to fit.

Get all three at the same time?

Hopefully hard because AES does not mix well with arithmetic mod n and n'.

Speed: on 64-bit machines, about 5 times faster than VSH, about 8 times slower than AES-256.

The future: new use cases for hash functions – an example: Isolated proofs of Knowledge

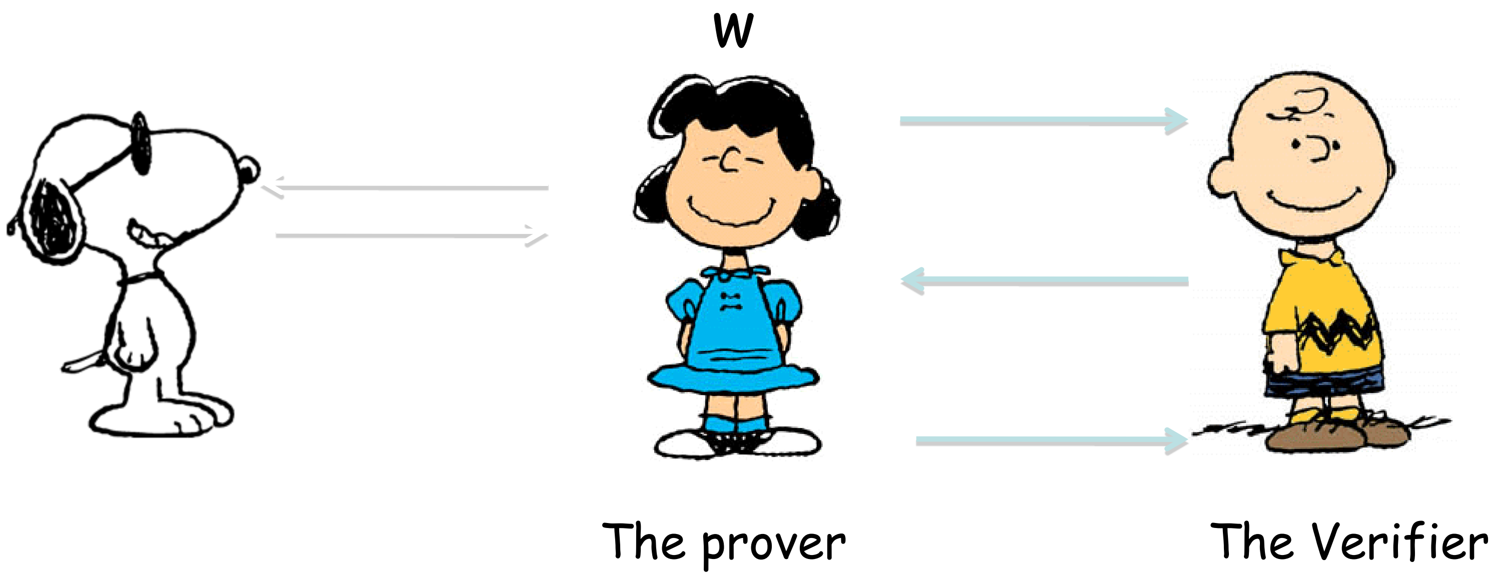


The prover claims to know a piece of information w

But in fact…

Some third party knows w and the prover is just relaying messages!

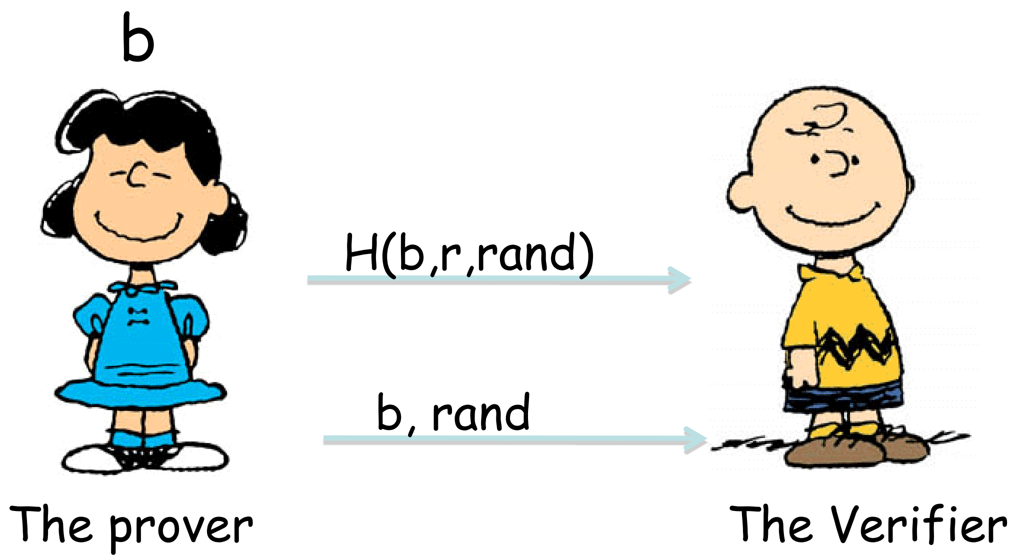
Avoiding the problem: limit the bandwith of the Prover's communication to third parties



Now the prover cannot just relay all messages. Can we design protocol such that the prover must know w to succeed?

[Damgård, Nielsen, Wichs EuroCrypt2008] Yes, and one solution follows if hash functions with certain properties exist..

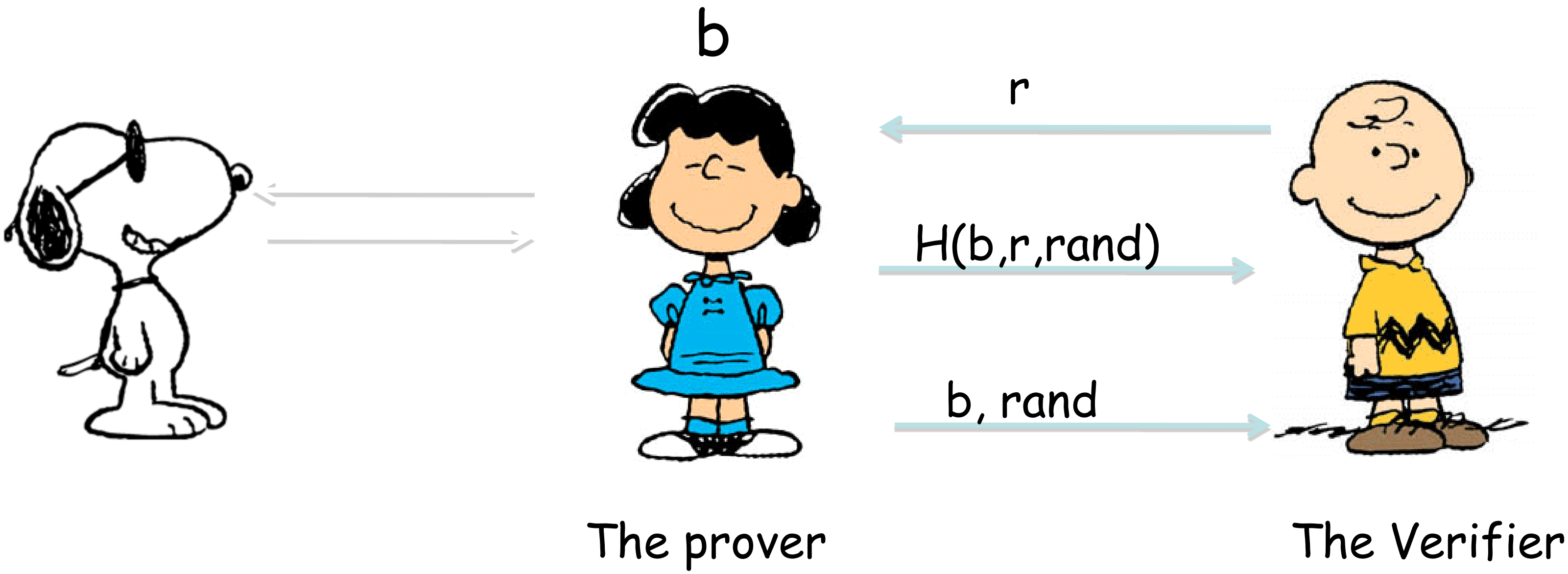
Towards a Solution: A bit commitment scheme based on hashing



Bit commitment scheme based on hash function H

Commit to bit b by sending H(b, randomness) to the verifier.
Open: reveal b and randomness – the verifier checks.
Binding: after commitment prover cannot change her mind (because H is collision intractable)
Hiding: Verifier cannot guess b before opening (because H compresses its input)

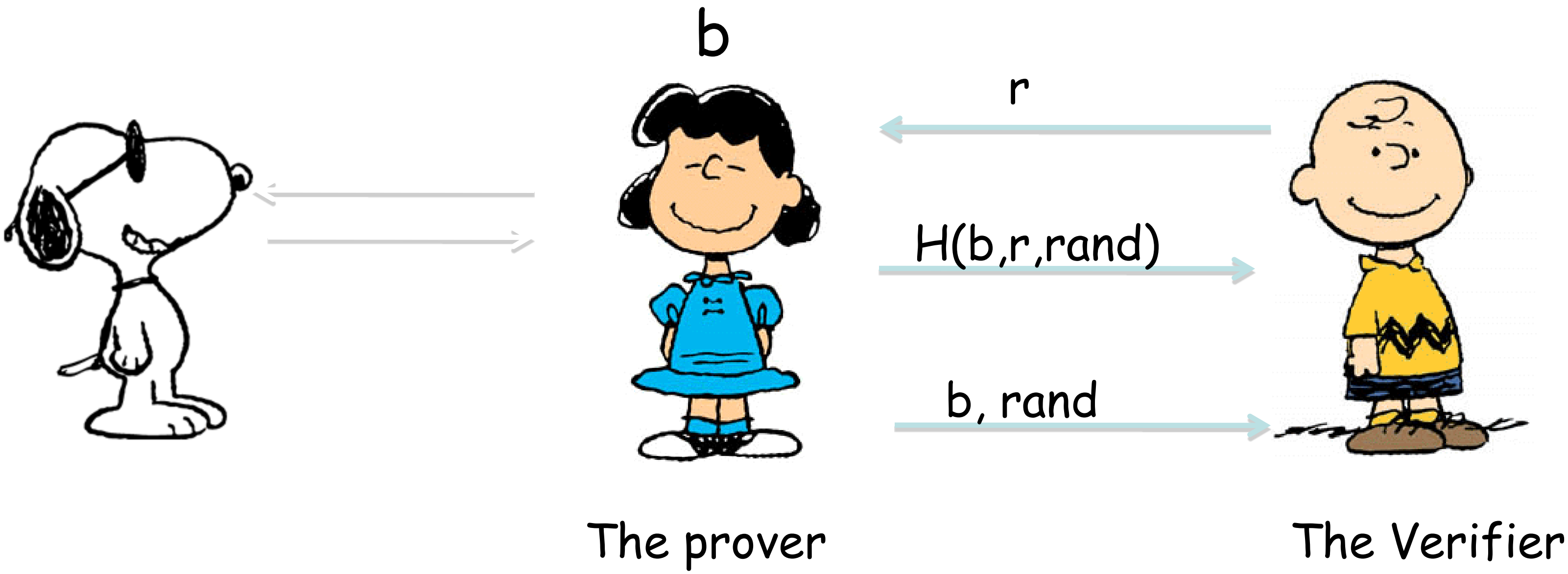
Solution: A bit commitment scheme with special properties is enough



The verifier sends a looong bit string r to the prover – so long that she cannot send full info on r to Snoopy.

Commit to bit b by sending H(b, r, randomness) to the verifier.
Open: reveal b and randomness – the verifier checks.
The special property: if the commitment can be opened, then the prover (and not Snoopy) must have known b already at commitment time.

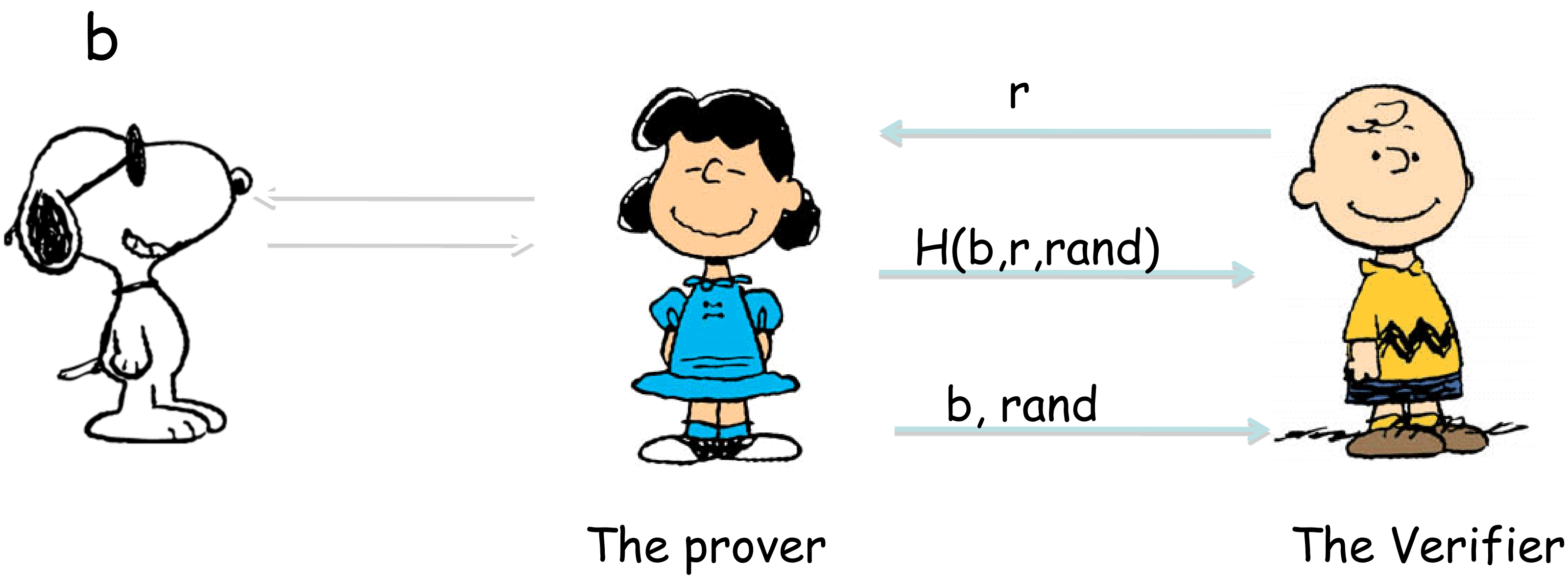
Does this work?



If H is a random oracle, then yes!

If the commitment can be opened, someone must have called the oracle with r as part of the input.
And (except with negligible probability) it was not Snoopy..
.. so the prover must have made the call and therefore knows b!

But we don't really need a random oracle..



What we really need is that H satisfies an extra assumption:

If Snoopy has input b and the prover has r, then any protocol that outputs H(b,r,rand) to the prover for some value of rand, without revealing b to the prover, must require more communication that the prover can do.

Open question: do hash functions with this property exist??

The End…

We have come a long way since the claw-free permutations of the 80-ties.

Simple collision intractability only is not enough these days..

Many new and interesting use cases for hash functions.

New designs and ideas wanted!