Can we Construct Unbounded Time-Stamping Schemes from Collision-Free Hash Functions

Margus Niitsoo

(joint work with Ahto Buldas)

October 5, 2008

What is timestamping

- What patent offices do
- Malicious patent clerk problem
- Information age sets new requirements

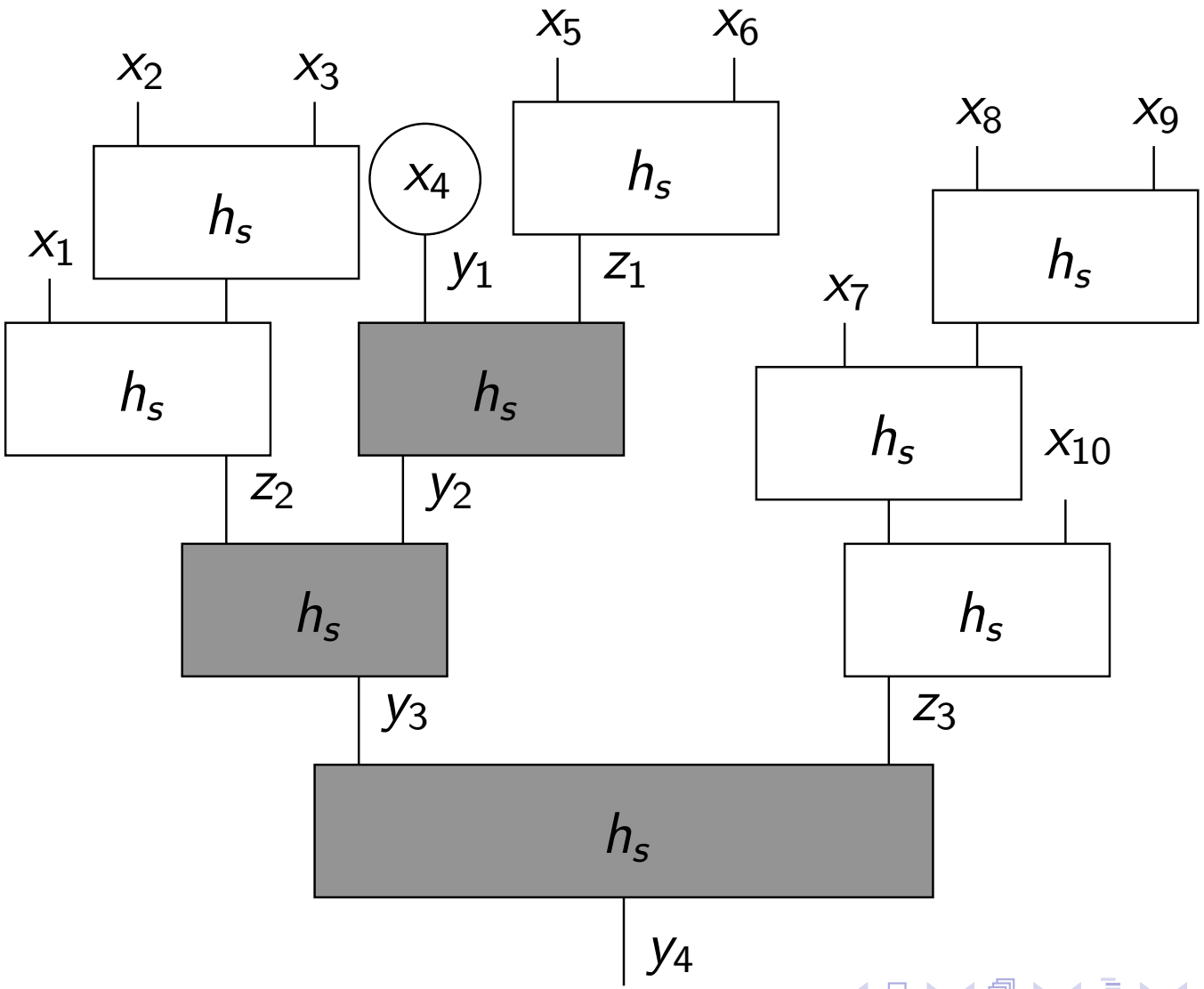
The Scheme of Harber and Stornetta

- Idea: publish a small hash value in the paper every day
- Fromally 3 parties: Client, Server, Repository
- Clients generate fixed sized signatures from their documents
- Server constructs a tree from signatures and uses a hash function $h_s : \{0,1\}^{2n} \longrightarrow \{0,1\}^n$ to calculate the root

The Scheme of Harber and Stornetta

- Idea: publish a small hash value in the paper every day
- Fromally 3 parties: Client, Server, Repository
- Clients generate fixed sized signatures from their documents
- Server constructs a tree from signatures and uses a hash function $h_s(x_1, x_2) = y$ to calculate the root

The Scheme of Harber and Stornetta



Collision resistance

- Intuitively, it should be hard to find two inputs $x \neq x'$ such that $h(x) = h(x')$.
- Such a pair is called a collision.
- It is assumed that collision-resistant functions do exist

The Security of the Scheme

- Security criterion – backdating a previously unknown document [BS04]

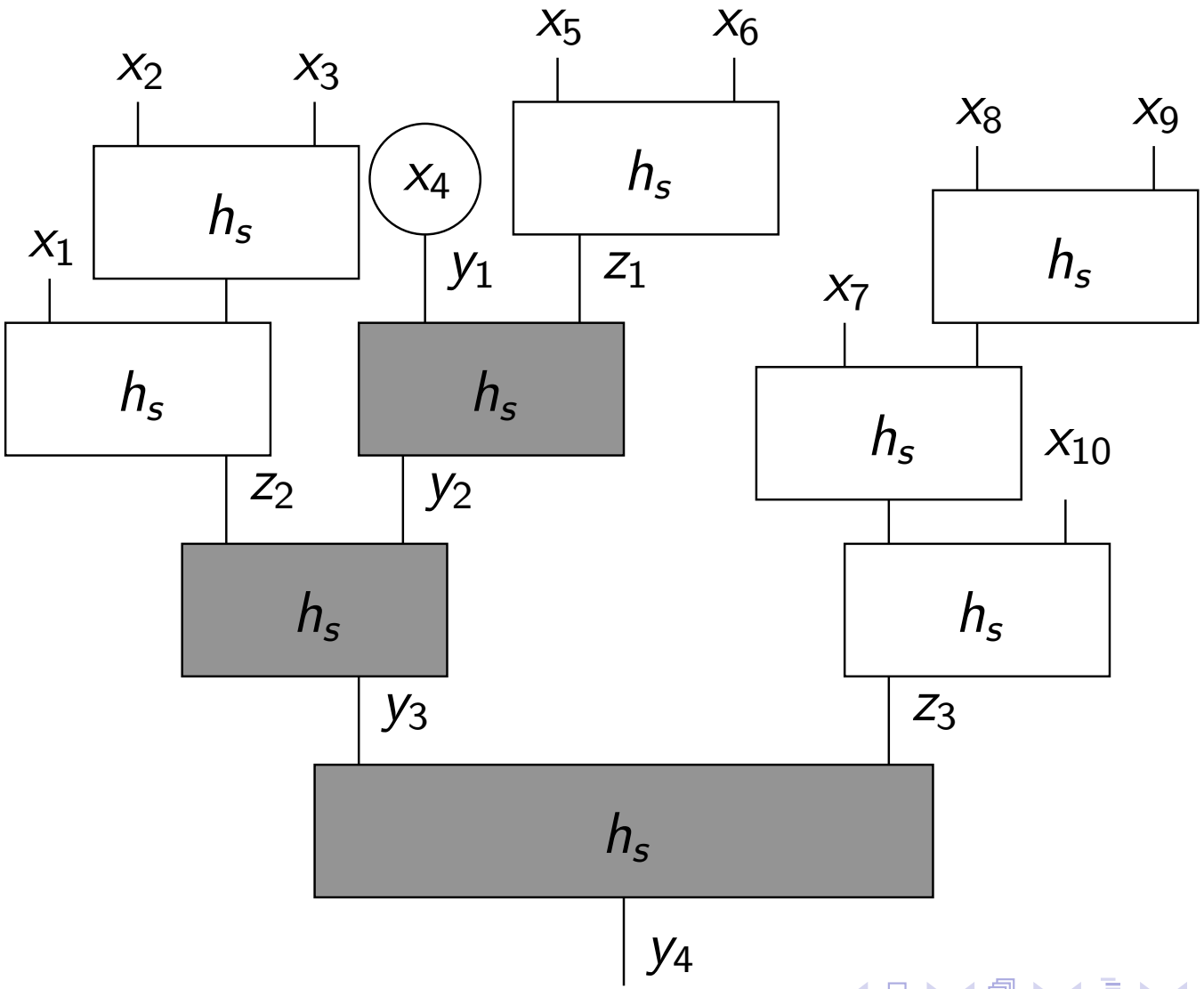
The Security of the Scheme

- Security criterion – backdating a previously unknown document [BS04]
- Bounded tree size and collision-resistance give a secure scheme [BL06]

The Security of the Scheme

- Security criterion – backdating a previously unknown document [BS04]
- Bounded tree size and collision-resistance give a secure scheme [BL06]
- For unbounded tree, collision-resistance gives nothing.
- We need something else!

Chain-resistance



Problem statement

- New criterion for Hash function security: Chain-resistance
- Independent of collision-resistance
- However, we may be able to construct chain-resistant hash functions from collision resistant ones
- We think it cannot be done and try to prove that

Reductions in cryptology

To show that one primitive can be constructed from another

- You need to show a construction that builds one from another
- You also need to prove that the new construction is secure
  - To do that, we show the contrapositive, that is, if it is not secure, then the original primitive used also is not.
  - That is done by constructing an explicit adversary

Example: Merkle-Damgård construction

How to construct a collision-resistant

$$h' : \{0,1\}^{4n} \longrightarrow \{0,1\}^{n}$$

from a collision-resistant

$$h : \{0,1\}^{2n} \longrightarrow \{0,1\}^{n}$$

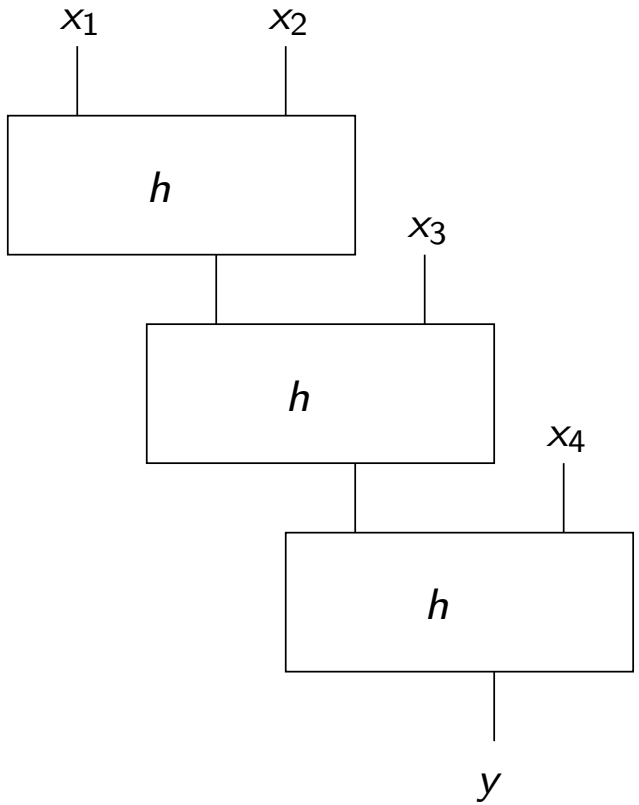
Example: Merkle-Damgård construction

How to construct a collision-resistant

$$h' : \{0,1\}^{4n} \longrightarrow \{0,1\}^n$$

from a collision-resistant

$$h : \{0,1\}^{2n} \longrightarrow \{0,1\}^n$$



Example: Merkle-Damgård construction

We assume we know how to break h'.
We want to know how to break h.

Breaking h' requires producing
$x = (x_1, x_2, x_3, x_4)$ and $x' = (x'_1, x'_2, x'_3, x'_4)$
so that $h'(x) = h'(x')$.

Breaking h requires producing
$z = (z_1, z_2)$ and $w = (w_1, w_2)$
so that $h(z) = h(w)$

Cryptographic reductions

- Let f be a collision-resistant hash function.
- To reduce chain-resistance to collision resistance we would need
  (a) An algorithmic construction of a secure chain-resistant function g based on f.
  (b) An explicit security reduction showing that if g is not chain-resistant then f cannot be collision resistant.
- (b) is stated in terms of adversaries
- This type of reduction is called fully black-box
- Vast majority of cryptographic reductions are of that type

Oracles

- An oracle is a black box that computes some specific function for us
- We can make calls to it, which are assumed to take unit time
- The function it computes may be hard or even impossible to compute in the real world
- Used extensively in complexity theory

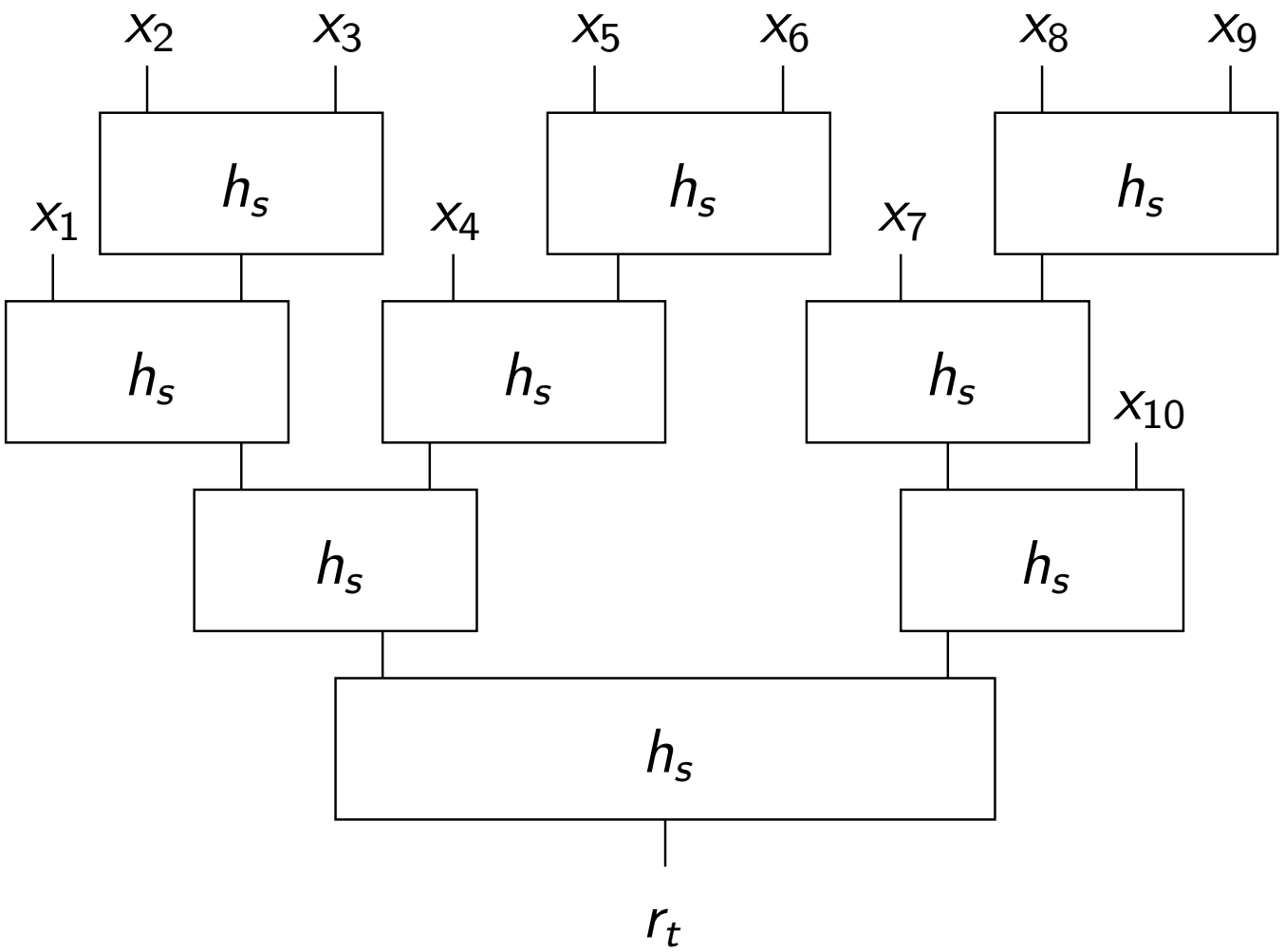
Oracle separation

- Since the constructions are algorithmic, they remain valid even if we allow the use of some oracle $\mathcal{O}$ inside f and the adversaries.
- Therefore, if we can find an oracle such that no function is chain-resistant but at least one function is collision-resistant, there cannot be a reduction.

Separation oracle

- We need an oracle such that
  - No function is chain-resistant
  - At least one function is collision-resistant
- For collision-resistance, just take a completely random function as one part of the oracle
- To break chain-resistance, take a function that creates a hash tree and gives certificates based on that

Adversary that constructs a tree



Separation oracle

- We need an oracle such that
  - No function is chain-resistant
  - At least one function is collision-resistant
- For collision-resistance, just take a completely random function as one part of the oracle
- To break chain-resistance, take a function that creates the full hash tree and gives certificates based on that

Separation oracle

- We need an oracle such that
  - No function is chain-resistant
  - At least one function is collision-resistant
- For collision-resistance, just take a completely random function as one part of the oracle
- To break chain-resistance, take a function that creates the full hash tree and gives certificates based on that
- That does not work! - The full tree oracle can be abused to find collisions

Separation oracle

- We need an oracle such that
  - No function is chain-resistant
  - At least one function is collision-resistant
- For collision-resistance, just take a completely random function as one part of the oracle
- To break chain-resistance, take a function that creates the full hash tree and gives certificates based on that
- That does not work! - The full tree oracle can be abused to find collisions
- It is enough to construct a partial tree if it is large enough
- We can choose which inputs we leave out

Our approach

- We give evidence in support of one oracle that should be suitable for the separation
- Namely, we try to find ways of abusing that oracle and then rule them out one by one

Quick recap

In short:

- We have two security conditions – chain and collision resistance

Quick recap

In short:

- We have two security conditions – chain and collision resistance
- We try to find an oracle that can always break one but not the other

Quick recap

In short:

- We have two security conditions – chain and collision resistance
- We try to find an oracle that can always break one but not the other
- We consider potential adversaries that could abuse that oracle to break collision-resistance

Quick recap

In short:

- We have two security conditions – chain and collision resistance
- We try to find an oracle that can always break one but not the other
- We consider potential adversaries that could abuse that oracle to break collision-resistance
- and prove that they cannot succeed given certain constraints

Adversary restrictions

- We fix one function h that we want to keep collision-resistant.
- We restrict the queries the adversary can make about h
  - Instead of having full access to h being broken for collision resistance, it is only allowed queries of type $h(x_1) = h(x_2)$
- Under these assumptions there exists a hash tree oracle that gives only a negligible increase to the probability of finding collisions.

Further generalizations

- The result can be extended to cover queries of type $h(x) = y$ and even $h(x_1) < h(x_2)$ and the result holds for the combination of the three as well.
- The oracle for $h(x) < y$ is equivalent to the full oracle for h but we cannot extend our approach to that

Possible changes to the oracle

- In general, this oracle is promising because we rule out the most simple ways of exploiting it

Possible changes to the oracle

- In general, this oracle is promising because we rule out the most simple ways of exploiting it
  - Ergo, the adversary either has to be really clever or really complex

Possible changes to the oracle

- In general, this oracle is promising because we rule out the most simple ways of exploiting it
  - Ergo, the adversary either has to be really clever or really complex
- Having a tree is actually a restriction – we could drop it

Possible changes to the oracle

- In general, this oracle is promising because we rule out the most simple ways of exploiting it
  - Ergo, the adversary either has to be really clever or really complex
- Having a tree is actually a restriction – we could drop it
  - However, restrictive models are simpler to work with

Possible changes to the oracle

- In general, this oracle is promising because we rule out the most simple ways of exploiting it
  - Ergo, the adversary either has to be really clever or really complex
- Having a tree is actually a restriction – we could drop it
  - However, restrictive models are simpler to work with
- We could choose the root randomly and try to construct a tree on top of that

Possible changes to the oracle

- In general, this oracle is promising because we rule out the most simple ways of exploiting it
  - Ergo, the adversary either has to be really clever or really complex
- Having a tree is actually a restriction – we could drop it
  - However, restrictive models are simpler to work with
- We could choose the root randomly and try to construct a tree on top of that
  - Would eliminate information leak from root but may introduce it into the certificates we give.

Conclusion

In the author's view

- We have given strong evidence that chain-resistant functions cannot be constructed from collision resistant ones

Conclusion

In the author's view

- We have given strong evidence that chain-resistant functions cannot be constructed from collision resistant ones
- The separation oracle proposed may give the required separation in the future

Conclusion

In the author's view

- We have given strong evidence that chain-resistant functions cannot be constructed from collision resistant ones
- The separation oracle proposed may give the required separation in the future
- proving the full separation probably needs a different approach or stronger mathematical methods

Thank You!

Thank you for attention! Any questions are welcome!