# Provably secure hash functions - do we care?

Krystian Matusiewicz

Technical University of Denmark

Quo Vadis 2008, 30 May 2008

## Problems and instances

- Problem – a general question to be answered, possesing some parameters and having a description of the expected solution.

  Example: Order elements of the array $A = [a_1, a_2, \ldots, a_n]$ in increasing order
- Instance – a particular case of the problem with all parameters fixed to specific values

  Example: Order array $[3, 6, 2, 4, 8, 9]$
- Algorithm – a sequence of operations that for any instance $I$ of the problem $P$ yields a solution of the problem.

  Example: Selection sort

## Instance size and time complexity

- Encoding scheme – a function that maps all problem instances to strings over a fixed alphabet

  Example: $\mathcal{A} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ;\}$, encoding of the array: $3; 6; 2; 4; 8; 9; ;$
- Instance size – number of symbols used to describe the instance using the selected encoding function
- Time complexity of the algorithm – expresses the number of operations needed by the algorithms as a function of input size for all problem instances

  Example: Selection sort uses at most $n$ swaps and $n(n-1)/2$ comparisons. Time complexity: $T(n) = n(n-1)/2 + n$.
- Polynomial-time algorithm – algorithm with time complexity bounded from above by a polynomial function.

## Easy and impossible problems

Polynomial-time algorithms are considered to be easy. Complexity class P: all algorithms that are solvable in polynomial-time.

However, there exist problems for which there is absolutely no algorithm that solves them...

- Halting problem: Given a description of a program and a finite input, decide whether the program finishes running or will run forever, given that input.
- Hilbert's tenth problem: Let $f \in \mathbb{Z}[x_1, \ldots, x_n]$. Is there $z \in \mathbb{Z}^n$ such that $f(z) = 0$ ?

## Anything in-between? Hard and harder problems

There are many problems that apparently require exponential time to solve them.

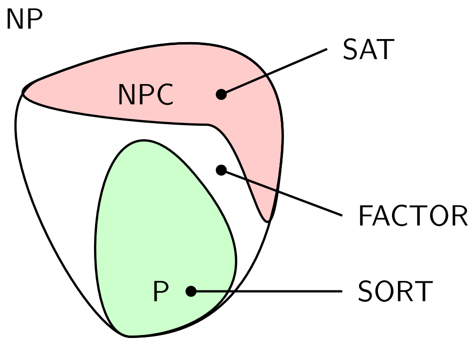
Example: SAT: Let $f \in \mathbb{F}_2[x_1, \ldots, x_n]$. Is there $z \in \mathbb{F}_2^n$ such that $f(z) = 0$ ?

Complexity class NP : problems solvable in polynomial time by non-deterministic algorithm. [Solutions can be verified by a polynomial time algorithm]

NP-Complete problems: Class of problems in NP that are "the hardest" problems in NP. [Any other problem in NP can be polynomially reduced to one in NP-C]

## Our view of complexity classes

We assume that $P \neq NP$



## Security proofs

We want to use difficult computational problems to our advantage.

If some computational problems seem to be hard and we want cryptosystems to be hard to break, maybe we can use intractable computational problems to construct cryptosystems.

Security reduction:

- If you can break the cryptosystem, you can solve this intractable problem.
- Since so many people have studied that hard problem (and other related ones), it is unlikely that there is an efficient method of solving it.

## Example 1: DLP-based hash function

- Proposed by Chaum, van Heijst, Pfitzman [CRYPTO'91]
- Let $(g_1, g_2, \ldots, g_t)$ be a sequence of randomly chosen generators of a cyclic group $\mathcal{G}$ of a prime order
- The function

$$F(x_1, \ldots, x_t) = g_1^{x_1} \cdot g_2^{x_2} \cdot \cdots \cdot g_t^{x_t}$$

  is collision-resistant provided that the discrete logarithm problem in the group $\mathcal{G}$ is hard.
- the group can be i.e. $\mathcal{G} = \mathbb{F}_{2^n}^*$ where $p = 2^n - 1$ is a Mersenne prime

## DLP-based hash function: security reduction

Let us focus on the smallest case

$$F(x, y) = g_1^x \cdot g_2^y$$

Assume that we have an algorithm that finds collisions in that function, i.e. finds pairs $(x_1, y_1)$, $(x_2, y_2)$. We have then

$$g_1^{x_1} \cdot g_2^{y_1} = g_1^{x_2} \cdot g_2^{y_2}$$

or

$$g_1^{x_1 - x_2} = g_2^{y_2 - y_1}$$

## DLP-based hash: security reduction

To solve DLP instance, we want to find such $\alpha$ that $g_2 = g_1^{\alpha}$.
From the previous equation:

$$g_1^{x_1 - x_2} = g_1^{\alpha(y_2 - y_1)}$$

But this gives us

$$x_1 - x_2 = \alpha(y_2 - y_1) \pmod{(\#\mathcal{G})}$$

and we can solve this for $\alpha$.

- This can be generalized for $t > 2$ by induction.

## Example 2: Hash function as hard as factoring

- Proposed by Damgård [EUROCRYPT'87]
- Let $N = pq$ where $p, q$ are primes $\equiv 3 \pmod 4$
- Compression function $h : \{0,1\} \times SQ(N) \to SQ(N)$ defined as
$$h(x,y) = a_x \cdot y^2 \pmod N$$
where $SQ(N)$ is the set of quadratic residues mod N and $a_0, a_1 \in SQ(N)$ are randomly chosen.

## Factoring-based hash f. : security reduction

If we have a collision

$$a_{x_1} \cdot y_1^2 = a_{x_2} \cdot y_2^2$$

it means that

$$a_{x_1} \cdot y_1^2 - a_{x_2} \cdot y_2^2 = N$$

and we can find a factor of $N$ with probability $1/2$ by examining

$$\gcd(a_{x_1} \cdot y_1^2 - a_{x_2} \cdot y_2^2, N)$$

## Example 3: Lattice-based hash function

- Proposed by Goldreich, Goldwasser, Halevi '96
- Let $q \in \mathbb{Z}$, and $A$ be an $n \times m$ matrix with entries from $\mathbb{Z}_q$.
- Let $x \in \{0,1\}^m$ be a sequence of zeros and ones, then we define

$$h(x) = A \cdot x$$

In other words,

$$h(x) = \sum_{i:x_i=1} a_i \pmod{q},$$

we sum all the columns of the matrix $A$ that correspond to ones in $x$.

## Lattice based hash function: Security reduction

- Finding collisions means finding two vectors $x, y \in \{0,1\}^m$ such that $A \cdot x = A \cdot y$
- Equivalent to finding a ternary vector $z \in \{-1,0,1\}$ such that $Az = 0$.
- Can be described as finding an integer vector s.t. $||z||_\infty = 1$ in the lattice spanned by $A$
- lattice Shortest Vector Problem – approximating SVP in any $L_p$ norm is NP-hard

## Problems with provably-secure constructions

- Speed
- Digest length
- Real-life security vs. theoretical security
- Structure

## Speed

- Computational problems have rich, complex structure: long integer arithmetic, matrix operations, finite field operations, elliptic curve operations etc.
- Modern processors are not optimized towards such tasks
- Efficiency dramatically worse that dedicated designs where the problem is tuned for the processor

Possible ways out:

- Processors evolve to include support for some cryptography-related operations
- Researchers come up with intractable problems suitable for fast implementations

## Attack cost vs. digest length

Consider an idealised hash function (i.e. modeled as random oracle) with $n$ bits of output size

- we need $2^n$ queries to find a preimage
- we need $2^{n/2}$ queries to find a collision

For dedicated designs, $2^{80}$ evaluations is thought to be out of reach for some time so the hash length can be $160$ bits.

## Digest lengths of constructions based on hard problems

Factoring and discrete logarithms in finite fields have subexponential complexity.

- **factoring:** In 2005 RSA-200 was factored (**663**-bit modulus), estimated work effort: 75 years on a single Opteron 2.2GHz
- **discrete log:** In 2001 discrete logs in $F_{2^{607}}$ (**607** bits) were possible on a fairly reasonable set of PCs

On the other hand, the biggest ECDLP challenge solved so far is **109**-bit [2004]

## Internal structure

- For an “ideal” hash function there is no structure – any two outputs are completely unrelated
- Functions based on algorithimc problems tend to have some residual structure