Introduction to

Cryptography

Lecture 16–18

©Michael Luttenberger

Chair for Foundations of Software Reliability and Theoretical Computer Science
Technische Universität München

2019/01/18, 16:35

1 Lecture 16 - 18 – OWF candidates, Construction of PRGs, HF

1 Lecture 16 - 18 – OWF candidates, Construction of PRGs, HF

- One-way functions
- Candidates for OWFs and OWPs
- One-way functions from computationally secret encryption*
- From one-way permutations to pseudorandom generators
- Hash functions

- Informally, a function $f\colon X \to Y$ is one-way if
  - it is “easy” to compute $f(x)$ for any $x \in X$, but
  - it is “infeasible” to compute $f^{-1}(y)$ for most $y \in Y$.
- In order to give a formal definition of “one-way” we need to make to precise what we mean by “easy”, “infeasible”, and “most”.
- Just as in the case of PRGs and PRFs/PRPs one has to use either concrete or asymptotic bounds.
- We only consider the asymptotic definition:
  - The (security) parameter becomes the problem length $n = |x|$ with $X_n := \{x \in X\colon\ |x| = n\}$.
  - “easy” becomes DPT-computable.
  - “infeasible” and “most” becomes that any PPT-adversary has only a negligible chance to compute any $x' \in f^{-1}(f(x))$ for $x \overset{u}{\in} X_n$.

- **Definition**: simple one-way function/permutation (OWF/OWP)

  A DPT-computable function

  $$f\colon \{0,1\}^* \to \{0,1\}^*$$

  is called a simple OWF if for any PPT-algorithm $\mathcal{A}$

  $$\varepsilon(n) := \Pr_{x \overset{u}{\in} \{0,1\}^n}\left[\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))\right]$$

  is negligible in n.

  It is a simple OWP if $f(\{0,1\}^n) = \{0,1\}^n$ for all $n \in \mathbb{N}$.

- **Theorem**: (w/o proof)

  If (simple) OWF exist, then $\mathbf{P} \neq \mathbf{NP}$.

▷ We can only conjecture that certain problems yield OWFs.

- As simple OWF map binary strings to binary strings,

  it is quite cumbersome to really paraphrase actual problems like

  - Factorization of integers:

    For a given $N \in \mathbb{N}$ find non-trivial x, y s.t. $N = xy$.

  - Discrete logarithm:

    For a cyclic group $\mathbb{G} = \langle g \rangle$ and $y \in \mathbb{G}$ find x s.t. $g^x = y$.

  as a simple OWF.

▷ For these problems, it is more convenient to use collections of OWFs.

- **Definition**: A PPT-function collection $\mathcal{F} = (\mathsf{Gen}, f)$ consists of

Algorithm	Type	Input	Output
Gen	PPT	1^n	$I \overset{r}{\in} \mathcal{I}_n$ with $\lvert I\rvert \geq n$
f	DPT	$I \in \mathcal{I}_n$, $x \in \mathsf{Dom}_I$	$y \in \mathsf{Rng}_I$

where

Gen generates function parameters $I \in \mathcal{I}_n$.

Every parameter I defines a domain Dom_I and range Rng_I, and the function $f_I \colon \mathsf{Dom}_I \to \mathsf{Rng}_I : x \mapsto f_I(x) := f(I, x)$

$\mathcal{F}$ is a permutation collection if f_I is a permutation on $\mathsf{Dom}_I = \mathsf{Rng}_I$ for every $I \in \mathcal{I}_n$.

- **Definition**: A one-way function/permutation (collection) (OWF/OWP) $\mathcal{F} = (\mathsf{Gen}, f, \mathsf{Smpl})$ is a function/permutation collection (Gen, f) plus a sampling algorithm

Algorithm	Type	Input	Output
Smpl	PPT	$I \in \mathcal{I}_n$	some x where $x \overset{r}{\in} \mathsf{Dom}_I$ except for negl. prob.

such that for any PPT-adversary $\Pr\left[\mathsf{Win}^{\mathrm{OWF}}_{n,\mathcal{F}}\right](\mathcal{A})$ is negl. in n:

1. Alice&Bob generate $I \overset{r}{:=} \mathsf{Gen}(1^n)$, choose $x \overset{r}{:=} \mathsf{Smpl}(I)$, compute $y := f_I(x)$, and pass (I, y) to Eve.
2. Eve runs $\mathcal{A}(I, y)$ to obtain x'.

▷ Let $\mathsf{Win}^{\mathrm{OWF}}_{n,\mathcal{F}}(\mathcal{A})$ be the event that $f_I(x') = y$.

Short: $\Pr_{I \overset{r}{:=} \mathsf{Gen}(1^n), x \overset{r}{:=} \mathsf{Smpl}(I)}\left[\mathcal{A}(I, f_I(x)) \in f_I^{-1}(f_I(x))\right]$ is negl. in n.

- Generic setup for factorization:
- Gen is deterministic with $I_n := \mathsf{Gen}(1^n)$
- $\mathsf{Dom}_{I_n} \subseteq [0, 2^{n/k} - 1]^k$ for some k determined by I_n
- $\mathsf{Rng}_{I_n} = [2^{n-1}, 2^n - 1]$
- $f_{I_n}(x_1, x_2, \ldots, x_{k_n}) := x_1 \cdot x_2 \cdots x_{k_n}$ for $(x_1, \ldots, x_k) \in \mathsf{Dom}_{I_n}$
- Smpl_{I_n} chooses $(x_1, x_2, \ldots, x_k) \overset{u}{\in} \mathsf{Dom}_{I_n}$

▷ Crucial point: choice of Dom_{I_n}.

- Generic setup for discrete logarithm:
- Gen outputs some description of a cyclic group $\mathbb{G} = \langle g \rangle$
  - Description: algorithm which allows to compute in $\langle \mathbb{G}, \cdot, 1 \rangle$ efficiently.
  - E.g. for $\mathbb{Z}_p^*$ knowledge of p suffices.
- $\mathsf{Dom}_{\mathbb{G}} = \mathbb{Z}_{|\mathbb{G}|}$
- $\mathsf{Rng}_{\mathbb{G}} = \mathbb{G}$
- $f_{\mathbb{G}}(x) := g^x$ computed in $\mathbb{G}$
- $\mathsf{Smpl}_{\mathbb{G}}$ chooses $x \overset{u}{\in} \mathbb{Z}_{|\mathbb{G}|}$.

▷ Crucial point: choice of $\mathbb{G}$

- Obviously, every simple OWF/OWP is also a OWF/OWP collection.
- We can transform any OWF/OWP collection into a simple OWF.

  Given $(\mathsf{Gen}, \mathsf{Smpl}, f)$, the single DPT-algorithm $\widetilde{f}$ treats its input $\{0,1\}^n$ as random bit string which it uses to run Gen and Smpl.

▷ "'$\widetilde{f}(x)$:= deduce n from $|x|$; split $x = x_{\mathsf{Gen}}||x_{\mathsf{Smpl}}$;
$I := \mathsf{Gen}(1^n, x_{\mathsf{Gen}})$; return $y := f_I(\mathsf{Smpl}(I, x_{\mathsf{Smpl}}))$"'

1. Lecture 16 - 18 – OWF candidates, Construction of PRGs, HF
   - One-way functions
   - Candidates for OWFs and OWPs
   - One-way functions from computationally secret encryption*
   - From one-way permutations to pseudorandom generators
   - Hash functions

- **Notation**:

  N positive interger ($N \in \mathbb{N}$), usually $N > 1$.

  N is an n-bit integer if $2^{n-1} \leq N < 2^n$.

  $d|N$ short for "d divides N".

  d is a nontrivial factor of N if $d|N$ and $d \notin \{1, N\}$.

- **Problem**: Integer factorization

  Given a positive integer N, find any nontrivial factor d of N – if there is one.

- **Example**: On input $N = 12345678910111213$, find 113.

- **Ex**: Let $\mathcal{A}$ be an algorithm which finds a nontrivial factor of a given N if there is one, and denote by $T_{\mathcal{A}}$ its running time.

  Show how to compute the complete prime factorization of an n-bit integer N in time $n \cdot T_{\mathcal{A}}(n)$.

- Generic setup for factorization (reminder):
- Gen is deterministic with $I_n := \mathsf{Gen}(1^n)$
- $\mathsf{Dom}_{I_n} \subseteq [0, 2^{n/k} - 1]^k$ for some k determined by I_n

  (i.e. a k-tuple of integers representable with n/k bits each)
- $\mathsf{Rng}_{I_n} = [0, 2^n - 1]$
- $f_{I_n}(x_1, x_2, \ldots, x_{k_n}) := x_1 \cdot x_2 \cdots x_{k_n}$ for $(x_1, \ldots, x_k) \in \mathsf{Dom}_{I_n}$
- Smpl_{I_n} chooses $(x_1, x_2, \ldots, x_k) \stackrel{u}{\in} \mathsf{Dom}_{I_n}$

▷ Crucial point: choice of Gen resp. Dom_{I_n}

  - Product yields hard to factorize number.
  - Has to be efficiently samplable.

▷ **Def.**: We say that factorization is hard w.r.t. Gen

  if (we conjecture that) above is a OWF for this particular Gen.

- How difficult is it to find a factor?

▷ For most N, it's trivial: assume we choose $N \overset{u}{\in} [0, 2^n - 1]$, then N is even with prob. $1/2$.

▷ **Ex**: For $d < 2^n$, give a lower bound on the prob. that $d|N$ when $N \overset{u}{\in} [0, 2^n - 1]$.

- **Conjecture**: Standard conjecture for factorization

  Factorization becomes an OWF if N is the product of two distinct random $n/2$-bit primes.

▷ I.e. $\text{Dom}_{I_n} = \{(p, q) \in [2^{n/2-1}, 2^{n/2} - 1]^2 \colon p, q \text{ prime} \wedge p \neq q\}$

▷ **Ex**: Assume p, q are generated using rejection sampling.

  Approximate the probability that $p = q$ using the prime number theorem.

- **Def**: We use $\mathsf{Gen}\mathbb{P}^2$ to denote a PPT-algorithm that outputs a pair of distinct $n/2$-bit primes chosen uniformly at random from the set of all pairs of distinct $n/2$-bit primes except for negligible probability w.r.t. n (e.g. by means of rejection sampling using Miller-Rabin as primality test).

▷ Slightly imprecsie the standard conjecture for factorization becomes

factorization is hard w.r.t. $\mathsf{Gen}\mathbb{P}^2$

($\mathsf{Gen}\mathbb{P}^2$ actually combines parameter generation and sampling.)

- **Remark**: Factorization is hard w.r.t. $\mathsf{Gen}\mathbb{P}^2$ iff computing square roots of quadratic residues modulo $N = pq$ is hard. (See the appendix for details.)

- Best algorithms known (aka. published) today:

▷ Classical computers:

General number field sieve factorizes N in time $\mathcal{O}(e^{(\frac{64}{9}n)^{\frac{1}{3}}(\log n)^{\frac{2}{3}}})$.

For $n = 1024$, this is roughly $c \cdot 2^{89}$ in the worst case for some constant c.

▷ Quantum computers:

Shor's algorithm runs in time $\mathcal{O}(n^3)$ and requires $\mathcal{O}(n)$ qubits.

  - Timeline of quantum computing

    (D-Wave's systems are currently not considered to be universal quantum computers.)

▷ See here for a list of "factorization records".

▷ So we essentially conjecture that $\mathrm{Gen}\mathbb{P}^2$ only produces worst-case problem instances for integer factorization.

- **Problem**: Discrete logarithm problem (DLP)

  Given a description of a finite cyclic group $\mathbb{G}$, a generator $\langle g \rangle = \mathbb{G}$, and a group element $y \in \mathbb{G}$, find an $x \in \mathbb{Z}$ such that $g^x = y$ in $\mathbb{G}$.

- **Example**: Let $(p, p-1, g)$ be a description of $\langle g \rangle = \mathbb{Z}_p^*$ for p prime.

  Given $p = 1019$, $g = 7$, and $y = 65$,

  find x with $7^x \equiv 65 \pmod{1019}$.

- Generic setup for discrete logarithm (reminder):
- Gen outputs some description of a cyclic group $\mathbb{G} = \langle g \rangle$
- $\mathsf{Dom}_{\mathbb{G}} = \mathbb{Z}_{|\mathbb{G}|}$
- $\mathsf{Rng}_{\mathbb{G}} = \mathbb{G}$
- $f_{\mathbb{G}}(x) := g^x$ computed in $\mathbb{G}$
- $\mathsf{Smpl}_{\mathbb{G}}$ chooses $x \overset{u}{\in} \mathbb{Z}_{|\mathbb{G}|}$.

▷ Crucial point: choice of Gen resp. $\mathbb{G}$
  - Above needs to become an OWF.
  - Need to be able to efficiently compute in $\mathbb{G}$.

- **Def.**: We say that the DLP is hard w.r.t. Gen

  if (we conjecture that) above is an OWF for this particular Gen.

- The parameters output by Gen is in the case of the DLP a description of a group.

▷ We could output a table of the group operation (Cayley table)

BUT as the adversary is given the parameter/descriptions this would make brute-force search feasible.

▷ The description has to include

  - the chosen generator g
  - the size/order of the group $|\mathbb{G}|$
  - enough information to efficiently compute in $\mathbb{G}$

- For instance, for $\mathbb{Z}_p^*$ the prime p is a succinct, but efficient description.

- As in the case of integer factorization:

  Want to use only groups for which the DLP is always hard.

- Computing the discrete logarithm is easy in $\mathbb{Z}_M \hat{=} \langle \mathbb{Z}_M, +, 0 \rangle$.

- As every cyclic group $\mathbb{G}$ is isomorpic to $\langle \mathbb{Z}_{|\mathbb{G}|}, +, 0 \rangle$, we want "worst-case" representations of $\mathbb{Z}_M$ which make computing the discrete logarithm hard.

- Let $M = |\mathbb{G}|$, and assume $M = p^r N$ with p prime and $\gcd(p, N) = 1$.

  By the CRT: $\langle g \rangle = \mathbb{G} \cong \mathbb{Z}_M \cong \mathbb{Z}_{p^r} \times \mathbb{Z}_N \cong \langle g^{p^r} \rangle \times \langle g^N \rangle$

  That is: we can remove small prime factors of M, and work in smaller subgroups of $\mathbb{G}$.

▷ For this reason, we want to use primes p such that $p - 1$ has one dominating prime factor, e.g. as in the case of a safe prime $p = 2q + 1$ (q also prime).

- Current conjectures which groups to use for a given n:
  - Pick a (safe) n-bit prime p and use $\mathbb{Z}_p^*$.
  - Pick a safe $n+1$-bit prime p and use $\mathbb{QR}_p$.

    Note: $\mathbb{QR}_p$ is of prime order q, i.e. we cannot use the CRT to move to smaller groups.
  - More general: strong primes

    A strong prime p is of the form $p = kq + 1$ with q an n-bit prime and k "small" so that we can efficiently determine k from $p - 1$.

    Let g generate $\mathbb{Z}_p^*$, and use $\langle g^k \rangle$ for $\mathbb{G}$.

    Ex: $\langle g^k \rangle$ is of prime order q, and a subgroup of $\mathbb{QR}_p$.
  - Cyclic subgroups of certain elliptic curves.

    For certain curves, only generic, i.e. exponential-time algorithms are known. Allows to resort to smaller groups which allow for more efficient computation.

- **Definition**:

  Let $\mathsf{Gen}\mathbb{Z}^*_{\mathsf{safe}}$ be a PPT-algorithm which, on input 1^n, generates

  (i) an n-bit Sophie-Germain prime q, so that $p = 2q + 1$ is a safe prime, and (ii) $\underline{\text{a generator } g \text{ of } \mathbb{Z}^*_p}$, and

  outputs $I = (p, p - 1, g)$ as description of $\langle g \rangle = \mathbb{Z}^*_p$.

▷ **Conjecture**: The DLP is hard w.r.t. $\mathsf{Gen}\mathbb{Z}^*_{\mathsf{safe}}$.

- **Definition**:

  Let $\mathsf{Gen}\mathbb{QR}_{\mathsf{safe}}$ be a PPT-algorithm which, on input 1^n, generates

  (i) an n-bit Sophie-Germain prime q, so that $p = 2q + 1$ is a safe prime, and (ii) $\underline{\text{a generator } g \text{ of } \mathbb{QR}_p}$, and

  outputs $I = (p, q, g)$ as description of $\langle g \rangle = \mathbb{QR}_p$.

▷ **Conjecture**: The DLP is hard w.r.t. $\mathsf{Gen}\mathbb{QR}_{\mathsf{safe}}$.

▷ **Remark**: W.r.t. to $\langle \mathbb{Z}_p^*, \cdot, 1 \rangle$ with p prime, the map

$f_{(p,p-1,g)} \colon \mathbb{Z}_{p-1} \to \mathbb{Z}_{p-1} \colon x \mapsto (g^x \bmod p) \bmod (p-1)$

is a permutation on $\mathbb{Z}_{p-1}$.

The conjecture that DLP is hard w.r.t. $\mathsf{Gen}\mathbb{Z}^*_{\mathsf{safe}}$ therefore yields a collection of one-way permutations (OWPs).

▷ **Ex**: Recall that

modulo a safe prime p, we have $(x^2)^{\frac{p+1}{4}} \equiv \pm x \pmod{p}$.

That is, we can efficiently map every $x^2 \in \mathbb{QR}_p$ to its positive square root in $\{1, \ldots, q\}$,

thereby turning the DLP w.r.t. $\mathsf{Gen}\mathbb{QR}_{\mathsf{safe}}$ into a OWP over $\mathbb{Z}_q$.

- ▷ Recall: Except for negl. probability, based on the conjecture by Hardy-Littlewood, we can generate both a (random) n-bit safe prime, and a generator of $\mathbb{Z}_p^*$ in time polynomial in n.
- In practice, the actual group (description) is sometimes simply chosen from a list of precomputed descriptions, in particular, when using subgroups of elliptic curves (see e.g. [here]).
- ▷ But see also this talk by Dan Bernstein why this is perhaps not the best way to use elliptic curves: [PDF]
- The main reason why Gen is a randomized algorithm is that this allows us to efficiently find by means of sampling the parameters for a group, e.g. some (random safe) n-bit prime and some (random) generator of $\mathbb{Z}_p^*$ resp. $\mathbb{QR}_p$.

- Best algorithms known today:

▷ Classical computers: Depends on $\mathbb{G}$.

  - If $\mathbb{G} \leq \mathbb{Z}_p^*$ modulo a prime p: General number field sieve can be adapted; super-polynomial, but subexponential running time in $|\mathbb{G}|$.
  - If $\mathbb{G} \leq \mathsf{GF}(2^n)$: Index calculus algorithm takes also super-polynomial, but subexponential time $|\mathbb{G}|$.
  - For a general cyclic group $\mathbb{G}$: Several generic algorithms are known (see here for a list), all of which run in exponential time $\mathcal{O}(\sqrt{|\mathbb{G}|})$ in the worst case.
  - **Remark**: A generic algorithm does not make use of the particular representation of $\mathbb{G}$ or the implementation of the group operation, and essentially treats the group as a black box. Generic algorithms cannot do better than $\mathcal{O}(\sqrt{|\mathbb{G}|})$ in the worst case [13].

▷ Quantum computers: Shor's algorithm can also be used.

- See here for a list of "DLP records".

- **Reminder**: Let $\mathbb{G}$ be a finite commutative group. Then:
- ▷ Its exponent $\lambda_{\mathbb{G}}$ is the least positive integer λ s.t. $\forall a \in \mathbb{G}\colon a^{\lambda} = 1$.
- ▷ If $\mathbb{G} = \mathbb{Z}_N^*$, then $\lambda(N) := \lambda_{\mathbb{Z}_N^*}$ is called the Carmichael function.
- ▷ Let $N = \prod_{i=1}^{r} p_i^{e_i}$ be a prime factorization of N.

  Then: $\lambda(N) = \mathrm{lcm}(\lambda(p_1^{e_1}), \ldots, \lambda(p_r^{e_r}))$

  where $\lambda(2) = 1$, $\lambda(4) = 2$, $\lambda(2^k) = 2^{k-2}$,

  and $\lambda(p^e) = (p-1)p^{e-1}$ for $p > 2$.
- ▷ The map $\exp_e\colon \mathbb{G} \to \mathbb{G}\colon x \mapsto x^e$ is a permutation iff $\gcd(e, \lambda_{\mathbb{G}}) = 1$.

  If $1 = \gcd(e, \lambda_{\mathbb{G}}) = ed + \lambda_{\mathbb{G}} f$, then $\exp_e^{-1} = \exp_d$.
  - In fact, $\exp_e$ is always a homorphism. So for $\gcd(e, \lambda_{\mathbb{G}}) = 1$ it is also an isomorphism.
- Just as for $\varphi(N)$, we do not know how to efficiently compute $\lambda(N)$ if factorizing N is hard.

- Reminder: $\varphi(N)$ vs. $\lambda(N)$

▷ **Ex**: Let $N = pq$ with p, q distinct primes. Then:

  - $\text{gcd}(e, \lambda(N)) = 1$ iff $\text{gcd}(e, \varphi(N)) = 1$.
  - $e \in \mathbb{Z}^*_{\lambda(N)} \Rightarrow \{e, e + \lambda(N)\} \subseteq \mathbb{Z}^*_{\varphi(N)}$.

▷ So, exp_e is also a bijection for $e \in \mathbb{Z}^*_{\varphi(N)}$, but we always can find distinct $e, e' \in \mathbb{Z}^*_{\varphi(N)}$ with $\text{exp}_e = \text{exp}_{e'}$.

▷ Example: Let $N = 11 \cdot 13$. Then $\varphi(N) = 120$ and $\lambda(N) = 60$.

Let $e = 61$. Then $\text{gcd}(e, \varphi(N)) = 1$, but $\text{Id} = \text{exp}_1 = \text{exp}_{61}$.

▷ **Ex**: $\text{exp}_e \neq \text{exp}_{e'}$ for distinct $e, e' \in \mathbb{Z}^*_{\lambda(N)}$.

▷ **Ex**: Let $N = 109 \cdot 163$.

For $e \in \mathbb{Z}^*_{\lambda(N)}$, how many $e' \in \mathbb{Z}^*_{\varphi(N)}$ are there with $\text{exp}_e = \text{exp}_{e'}$?

- The basic idea of the RSA problem is to use $\exp_e$ as a one-way function.

▷ In order to be able to compute $\exp_e$ we need to know

1. A (succinct) description of $\mathbb{G}$ which enables us to compute efficiently within $\mathbb{G}$.
2. The exponent $e \in \mathbb{Z}$. Wlog. $e \in \mathbb{Z}_\lambda$.

▷ $\exp_e$ can then be computed efficiently by means of repeated squaring.

- Necessary: The description must not allow to efficiently compute $\lambda_{\mathbb{G}}$.
  - Otherwise, compute $\lambda_{\mathbb{G}}$, then use EEA.
- Conjetured candidates for such groups:

  $\mathbb{G} = \mathbb{Z}_N^*$ with N a hard-to-factorize composite,

  e.g. let N be the product of two distinct $n/2$-bit primes,

  i.e. $N = pq$ with $(p, q) \stackrel{r}{:=} \mathsf{Gen}\mathbb{P}^2(1^n)$.

- **Definition**: Let

  Gen: on input 1^n, run $\mathsf{Gen}\mathbb{P}^2(1^n)$ to obtain p, q, set $N := pq$, compute $\lambda := \lambda(N)$, choose any $e \in \mathbb{Z}_\lambda^* \setminus \{1\}$, and output $I = (N, e)$.

  Smpl: on input $I = (N, e)$, output $x \overset{u}{\in} \mathbb{Z}_N^*$.

  f: on input $I = (N, e)$ and $x \in \mathbb{Z}_N^*$, output $f_I(x) := x^e \bmod N$.

  The RSA problem is hard w.r.t. $\mathsf{Gen}\mathbb{P}^2$ if above is a OWP.

- **Conjecture**:

  If factorization is hard w.r.t. $\mathsf{Gen}\mathbb{P}^2$, then RSA is hard w.r.t. $\mathsf{Gen}\mathbb{P}^2$.

- In fact, the RSA problem is a candidate for a trapdoor one-way permutation:

  When the trapdoor $\lambda(N)$ (or $\varphi(N)$ or p, q) is known, we can compute d such that $ed \equiv 1 \pmod{\lambda}$, and, hence, $(x^e)^d \equiv x \pmod{N}$.

- **Definition**: A trapdoor one-way permutation (TDP) $\mathcal{F} = (\mathsf{Gen}, f, \mathsf{Smpl})$:

Algorithm	Type	Input	Output
Gen	PPT	1^n	$(I, \mathsf{td}) \overset{r}{\in} \mathcal{I}_n \times \mathcal{T}_n$ with $\lvert I \rvert \geq n$
f	DPT	$I \in \mathcal{I}_n$, $x \in \mathsf{Dom}_I$	$y \in \mathsf{Rng}_I$
Smpl	PPT	$I \in \mathcal{I}_n$	$x \overset{r}{\in} \mathsf{Dom}_I$

such that (i) (I, td) allows to efficiently compute f_I^{-1},

but (ii) for any PPT-adversary $\Pr\left[\mathsf{Win}_{n,\mathcal{F}}^{\mathrm{TDP}}\right](\mathcal{A})$ is negl. in n:

1. Alice&Bob generate $(I, \mathsf{td}) \overset{r}{:=} \mathsf{Gen}(1^n)$ and destroy td, choose $x \overset{r}{:=} \mathsf{Smpl}(I)$, compute $y := f_I(x)$, and pass (I, y) to Eve.
2. Eve runs $\mathcal{A}(I, y)$ to obtain x'.

▷ Let $\mathsf{Win}_{n,\mathcal{F}}^{\mathrm{TDP}}(\mathcal{A})$ be the event that $x = x'$.

- **Lemma**: If Eve, given (N, e), can efficiently compute ...
  - p, q, she can efficiently compute $\varphi(N)$, $\lambda(N)$, and d.
  - $\varphi(N)$, she can efficiently compute p, q. (**Ex**)

    Hint: Show that $q^2 + q(N + 1 - \varphi(N)) + N = 0$ has to hold.
  - $\lambda(N)$, she can efficiently compute p, q. See, e.g., [6] p.232.
  - d, she can efficiently compute p, q. See, e.g., [5] p.143.
  - an $x \in \mathbb{Z}_N \setminus \mathbb{Z}_N^*$, she can efficiently compute p, q. (**Ex**)

▷ So, if the RSA problem is an OWP w.r.t the specific $\mathsf{Gen}\mathbb{P}^2$, none of the above can be done efficiently, in particular, factorizing N given (N, e) has to be hard.

- But: In general, it is not known, if solely the conjecture that factorizing N on input (N, e) is hard, suffices for the RSA problem to be an OWP. Only for the restricted setting of generic algorithms, this has been shown so far [1].

1 Lecture 16 - 18 – OWF candidates, Construction of PRGs, HF

- **Lemma**: Let $\mathcal{E} = (\mathsf{Gen}_{\mathcal{E}}, \mathsf{Enc}, \mathsf{Dec})$ be a deterministic comp. secret ES with $\mathsf{Gen}_{\mathcal{E}}(1^n) \overset{u}{\in} \mathcal{K}_n = \{0,1\}^n$ and $\{0,1\}^{2n} \subseteq \mathcal{M}_n$. Then the following $\mathcal{F} = (\mathsf{Gen}_{\mathcal{F}}, \mathsf{Smpl}, f)$ is a OWF:

  $\mathsf{Gen}_{\mathcal{F}}$: on input 1^n, output $I = m$ where $m \overset{u}{\in} \{0,1\}^{2n}$, $\mathsf{Dom}_m = \mathcal{K}_n$, and $\mathsf{Rng}_m = \mathcal{C}_n$.

  Smpl: on input $I = m$, output $k \overset{u}{\in} \{0,1\}^n$.

  f: on input $I = m$ and $k \in \{0,1\}^n$, output $f_m(k) := \mathsf{Enc}_k(m)$.

- ▷ **Remark**: As for comp. secrecy we only have to encrypt a single message, we can make the coin tosses ρ by Enc external, and simply supply Enc instead with the extended key $k||\rho$.

  For similar reasons, we can assume that $\mathsf{Gen}_{\mathcal{E}}(1^n)$ always generates a random key chosen uniformly from $\{0,1\}^n$.

  Then above statement says that it is has to be hard to find $k||\rho$ even when m and $c = \mathsf{Enc}_{k||\rho}(m)$ are known.

- Proof: Let $\mathcal{B}$ be any PPT-algorithm which tries to invert $\mathcal{F}$, i.e.

  on input $I = m$ and $c = \mathsf{Enc}_k(m)$, $\mathcal{B}$ tries to find some key in $f_I^{-1}(c) = \{k' \in \{0,1\}^n \mid \mathsf{Enc}_{k'}(m) = c\}$.

  We construct from $\mathcal{B}$ the following PPT-adversary $\mathcal{A}$ for the game $\textsc{IndEd}$ vs. $\mathcal{E}$:

Alice&Bob	$\mathcal{A}$	$\mathcal{B}$
run $\mathcal{A}(1^n)$		
	$m_0, m_1 \overset{u}{\in} \{0,1\}^{2n}$	
	return m_0, m_1	
$b \overset{u}{\in} \{0,1\}$		
$k \overset{u}{\in} \{0,1\}^n$		
$c := \mathsf{Enc}_k(m_b)$		
run $\mathcal{A}(1^n, \mathsf{Enc}_k(m_b))$		
	run $\mathcal{B}(m_1, \mathsf{Enc}_k(m_b))$	
		return k'
	if $\mathsf{Enc}_{k'}(m_1) = c$: return $r := 1$	
	else: return $r \overset{u}{\in} \{0,1\}$	

- Case $b = 1$:

Alice&Bob	$\mathcal{A}$	$\mathcal{B}$
run $\mathcal{A}(1^n)$		
	$m_0, m_1 \overset{u}{\in} \{0,1\}^{2n}$	
	return m_0, m_1	
$\cancel{b \overset{u}{\in} \{0,1\}}$ $b := 1$		
$k \overset{u}{\in} \{0,1\}^n$		
$c := \mathsf{Enc}_k(m_1)$		
run $\mathcal{A}(1^n, c)$		
	run $\mathcal{B}(m_1, c)$	
		return k'
	if $\mathsf{Enc}_{k'}(m_1) = c$: return $r := 1$	
	else: return $r \overset{u}{\in} \{0,1\}$	

▷ $\mathcal{A}$ wins iff $r = 1$.

▷ m_0 can be removed.

▷ Rearrange interaction into the game OWF.

- Case $b = 1$: From $\mathcal{B}$'s point of view

Alice&Bob&$\mathcal{A}$	$\mathcal{B}$
$m_1 \overset{u}{\in} \{0,1\}^{2n}$	
$k \overset{u}{\in} \{0,1\}^{n}$	
$c := \mathsf{Enc}_k(m_1)$	
run $\mathcal{B}(m_1, c)$	
	return k'
if $\mathsf{Enc}_{k'}(m_1) = c$: $\mathsf{Win}^{\text{IndEd}}_{n,\mathcal{E}}(\mathcal{A})$	
else: $\mathsf{Win}^{\text{IndEd}}_{n,\mathcal{E}}(\mathcal{A})$ with prob. $1/2$	

▷ $\mathcal{A}$ wins iff either

(i) $\mathcal{B}$ wins the game OWF vs. $\mathcal{F}$ or

(ii) $\mathcal{B}$ loses the game OWF vs. $\mathcal{F}$ but $\mathcal{A}$ guesses b correctly:

$$\Pr_{b=1}\left[\mathsf{Win}^{\text{IndEd}}_{n,\mathcal{E}}(\mathcal{A})\right] = \Pr\left[\mathsf{Win}^{\mathrm{OWF}}_{n,\mathcal{F}}(\mathcal{B})\right] + \left(1 - \Pr\left[\mathsf{Win}^{\mathrm{OWF}}_{n,\mathcal{F}}(\mathcal{B})\right]\right) \cdot \frac{1}{2}$$

- Case $b = 0$:

Alice&Bob	$\mathcal{A}$	$\mathcal{B}$
run $\mathcal{A}(1^n)$		
	$m_0, m_1 \overset{u}{\in} \{0,1\}^{2n}$	
	return m_0, m_1	
$\cancel{b \overset{u}{\in} \{0,1\}}$ $b := 0$		
$k \overset{u}{\in} \{0,1\}^n$		
$c := \mathsf{Enc}_k(m_0)$		
run $\mathcal{A}(1^n, c)$		
	run $\mathcal{B}(m_1, c)$	
		return k'
	if $\mathsf{Enc}_{k'}(m_1) = c$: return $r := 1$	
	else: return $r \overset{u}{\in} \{0,1\}$	

- ▷ $\mathcal{A}$ wins iff $r = 0$.
- ▷ Again, collapse Alice&Bob and $\mathcal{A}$.

- Case $b = 0$: From $\mathcal{B}$'s perspective:

Alice&Bob&$\mathcal{A}$	$\mathcal{B}$
$m_0 \overset{u}{\in} \{0,1\}^{2n}$	
$k \overset{u}{\in} \{0,1\}^{n}$	
$c := \mathsf{Enc}_k(m_0)$	
$m_1 \overset{u}{\in} \{0,1\}^{2n}$	
run $\mathcal{B}(m_1, c)$	
	return k'
if $\mathsf{Enc}_{k'}(m_1) = c$: $\neg\mathsf{Win}^{\textsc{IndEd}}_{n,\mathcal{E}}(\mathcal{A})$	
else: $\mathsf{Win}^{\textsc{IndEd}}_{n,\mathcal{E}}(\mathcal{A})$ with prob. $1/2$	

- $\mathcal{A}$ wins iff $\mathcal{B}$, on input (m_1, c) does not find some $k' \in \{0,1\}^n$ with $\mathsf{Enc}_{k'}(m_1) = c$ where $c = \mathsf{Enc}_k(m_0)$,
  - $\mathcal{B}$ can only find such a k' if $m_1 \in D_c = \{\mathsf{Dec}_{k''}(c) \mid k'' \in \{0,1\}^n\}$.

  ▷ As $m_1 \overset{u}{\in} \{0,1\}^{2n}$ and independently of m_0, the prob. for $m_1 \in D_c$ is $|D_c|\, 2^{-2n} \leq 2^{-n}$ for any ciphertext c.

and $\mathcal{A}$ guesses correctly: $\Pr_{b=0}\left[\mathsf{Win}^{\textsc{IndEd}}_{n,\mathcal{E}}(\mathcal{A})\right] \geq (1 - 2^{-n}) \cdot \frac{1}{2}$.

- In total:

$$4 \cdot \Pr\left[\mathsf{Win}^{\textsc{IndEd}}_{n,\mathcal{E}}(\mathcal{A})\right] \geq$$
$$2 \cdot \Pr\left[\mathsf{Win}^{\mathrm{OWF}}_{n,\mathcal{F}}(\mathcal{B})\right] + (1 - \Pr\left[\mathsf{Win}^{\mathrm{OWF}}_{n,\mathcal{F}}(\mathcal{B})\right]) + (1 - 2^{-n})$$

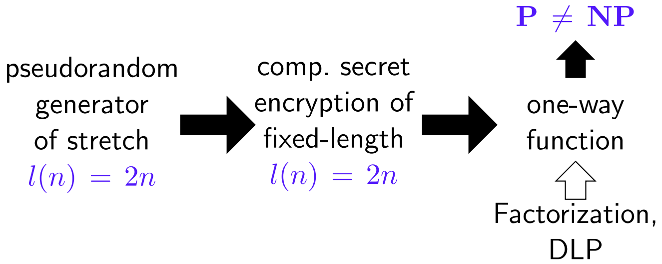
▷ Thus:

$$4 \cdot \left|\Pr\left[\mathsf{Win}^{\textsc{IndEd}}_{n,\mathcal{E}}(\mathcal{A})\right] - \tfrac{1}{2}\right| + 2^{-n} \geq \Pr\left[\mathsf{Win}^{\mathrm{OWF}}_{n,\mathcal{F}} \mathcal{B}\right].$$

▷ As $\mathcal{E}$ is comp. secret, the advantage of $\mathcal{A}$ is negl. in n, and, thus, any $\mathcal{B}$ can only succeed with negl. prob.

1 Lecture 16 - 18 – OWF candidates, Construction of PRGs, HF

- One-way functions
- Candidates for OWFs and OWPs
- One-way functions from computationally secret encryption*
- From one-way permutations to pseudorandom generators
- Hash functions



- What remains is to show that from OWFs we can also construct PRGs of stretch $l(n) = 2n$.
- We only discuss how PRGs of arbitrary (polynomial) stretch can be constructed from OWPs based on the idea of the Blum-Micali PRG.

▷ See [8] for a general proof based on any OWF.

- Recall:

  A DPT-computable function $G : \{0,1\}^* \to \{0,1\}^*$ which stretches inputs of length n to outputs of length $l(n) > n$ is a pseudorandom generator (PRG) if for every PPT-distinguisher $\mathcal{D}$

  $$\left|\Pr_{x \stackrel{u}{\in} \{0,1\}^n}[\mathcal{D}(G(x)) = 1] - \Pr_{y \stackrel{u}{\in} \{0,1\}^{l(n)}}[\mathcal{D}(y) = 1]\right| \text{ is negligible.}$$

- We follow the presentation of [2]:

  1. Yao's characterization of PRGs via unpredictability: $G(\cdot)$ is a PRG iff given the first i bits of $G(x)$ the $i+1$-th bit cannot be predicted (=computed) reliably.
  2. Hard-to-predict bits (hard-core predicates) hc_I for one-way permutations f_I allow to stretch a random string by one.
  3. Blum-Micali construction for obtaining arbitrary polynomial stretch.

     Repeat: output $\mathsf{hc}_I(x)$ (with x the seed) and "reseed" $x := f_I(x)$.

- **Definition**: A DPT-computable function $G : \{0,1\}^* \to \{0,1\}^*$ with polynomial stretch $l(n) \geq n$, i.e., $|G(x)| = l(|x|) \geq |x|$ for all x, is unpredictable (from the left) if for every PPT-algorithm $\mathcal{P}$ the prob.

$$\left|\Pr_{x,y=G(x),i}[\mathcal{P}(1^n, y_1 y_2 \ldots y_{i-1}) = y_i] - 1/2\right| .$$

is negligible for $x \overset{u}{\in} \{0,1\}^n$ and $i \overset{u}{\in} [l(n)]$ (and the coin tosses of $\mathcal{P}$).

- **Ex**: If $G(\{0,1\}^n) = \{0,1\}^n$ for every n, then it is unpredictable.
- **Ex**: Every PRG is unpredictable.
- **Ex**: Reformulate above definition as a game between Alice&Bob (using G) and Eve (using $\mathcal{P}$).

- **Theorem** [15]: (see the appendix for a proof)

  Let $G(\cdot)$ be as above. If G is unpredictable, then it is a PRG.

▷ Proof idea: Given a distinguisher $\mathcal{D}$ for stretch $s(n)$, define $\mathcal{P}_{\mathcal{D}}$ by:

  - Input: $y_1 y_2 \ldots y_{i-1}$
  - Set $y' := y_1 y_2 \ldots y_{i-1} y'_i \ldots y'_{s(n)}$ with $y'_j \overset{u}{\in} \{0,1\}$.
  - Return y_i iff $\mathcal{D}(1^n, y') = 1$ else return $1 - y_i$.

  That is, $\mathcal{P}$ guesses the missing bits in order to run $\mathcal{D}$, and assumes:

  $y_i = y'_i$ iff $\mathcal{D}$ thinks that y' has been generated by G.

- First goal: obtain a PRG of stretch $l(n) = n + 1$.

▷ Recall: Any DPT-computable f with $f(\{0,1\}^n) = \{0,1\}^n$ for all $n \in \mathbb{N}$ is unpredictable.

- Ansatz: $G(x) = f(x) || \mathsf{hc}(x)$
  - $\mathsf{hc}(x)$ is the single additional bit output by G.
  - It has to depend on the input x, so it needs to be some DPT-computable function from x to $\{0,1\}$.

▷ Yao's theorem: suffices to show that G is unpredictable.

▷ The first n bits from the left are unpredictable

as with $x \overset{u}{\in} \{0,1\}^n$ also $f(x) \overset{u}{\in} \{0,1\}^n$.

▷ So, predicting $\mathsf{hc}(x)$ when given $f(x)$ needs to be hard.

▷ If such an $\mathsf{hc}(\cdot)$ exists, it is called a hard-core predicate of f.

- **Definition**:

  A DPT-computable function hc is a hard-core predicate of a function $f\colon \{0,1\}^* \to \{0,1\}^*$ if for every PPT-algorithm $\mathcal{A}$ the prob.

  $$\left|\Pr_{x \overset{u}{\in} \{0,1\}^n}[\mathcal{A}(f(x)) = \mathsf{hc}(x)] - 1/2\right| \text{ is negligible.}$$

  Analogously, for a function collection: Then $I :\overset{r}{=} \mathsf{Gen}(1^n)$, $x :\overset{r}{=} \mathsf{Smpl}(1^n)$, and both $\mathcal{A}$ and hc are also given the parameter I.

- **Corollary**:

  If f is a PPT-computable permutation on $\{0,1\}^n$ (for every n) with hard-core predicate hc, then $G(x) := f(x)||\mathsf{hc}(x)$ is PRG of stretch $l(n) = n + 1$.

▷ Which functions possess hard-core predicates?

  Ex: f has to be OWP in order to possess a hard-core predicate.

- Any simple OWP can be transformed into a new simple OWP which has a hard-core predicate:
- **Theorem** [7]: (see the appendix for a proof)

  Let $f : \{0,1\}^* \to \{0,1\}^*$ be a simple one-way permutation.

  For every $n \in \mathbb{N}$ and $x, r \in \{0,1\}^n$ set

  $$g(x,r) := f(x)||r \text{ and } \mathsf{gl}(x,r) := \sum_{i=1}^{n} x_i \cdot r_i \mod 2.$$

  Then gl is a hard-core predicate of g.
- **Ex**: Show that $g(x,r)$ is also a OWP.
- Note that the adversary is given $f(x)||r$, so he knows r.

▷ Basic idea: As $r \overset{u}{\in} \{0,1\}^n$, to compute $\mathsf{gl}(x,r)$ the adversary has to be able to compute at least the majority of the linear combinations of the bits, which then suffices to compute x itself.

- The Goldreich-Levin predicate is quite inefficient, as it requires n additional truly random bits.
- For the conjectured OWP collections we have seen so far specific, practical hard-core predicates are known:
- DLP w.r.t. $\mathbb{Z}_p^*$ for p prime:

  $\mathsf{hc}_{(p,p-1,g)}(x) = (x < \frac{p-1}{2} \,?\, 1 : 0)$. [4]
- RSA:

  Any single bit of x, given $x^e \bmod N$, is as hard to compute as x itself. [10].

- **Theorem**:

  Let $f : \{0,1\}^* \to \{0,1\}^*$ be a permutation on $\{0,1\}^n$ for every n with hard-core predicate hc. For every $j \geq 0$ set

  $$\mathsf{BM}^j(x) := \mathsf{hc}(f^{j-1}(x))||\mathsf{hc}(f^{j-2}(x))||\dots||\mathsf{hc}(f(x))||\mathsf{hc}(x).$$

  Then $\mathsf{BM}^{l(|x|)}(x)$ is a PRG for every polynomial $l(n) > n$.

- **Corollary**: (**Ex**)

  $\mathsf{MB}(x, 1^s) := \mathsf{hc}(x)||\mathsf{hc}(f(x))||\dots||\mathsf{hc}(f^{s-1}(x))$ is a vlPRG.

- In pseudocode:
  - Input: $x \overset{u}{\in} \{0,1\}^*$, $j \in \mathbb{N}$.
  - for i from 1 to j:
    - output $\mathsf{hc}(x)$
    - $x := f(x)$

- **Remark**: The result holds analogously for a permutation collection $\mathcal{F}$ which has a hard-core predicate.

  Simply replace $f(x)$ by $f_I(x)$ and $\mathsf{hc}(x)$ by $\mathsf{hc}_I(x) = \mathsf{hc}(I, x)$ for $x \in \mathsf{Dom}_I$.

▷ E.g. for the DLP-OWP (known as the Blum-Micali PRG):

  - Given n, compute an n-bit prime p, and a generator g of $\mathbb{Z}_p^*$.

  ▷ Parameters: $I = (p, p-1, g)$.

  ▷ Hard-core predicate: $\mathsf{hc}_I(x) := (x < \frac{p-1}{2} ? 0 : 1)$.

  ▷ Note: As p is prime, $x \overset{u}{\in} \mathbb{Z}_{p-1}$ and $x \overset{u}{\in} \mathbb{Z}_p^*$ are equivalent.

- We want to show that the Blum-Micali construction is a PRG of variable stretch.

▷ We only need to show that $\mathsf{BM}^{l(n)}(\cdot)$ is a PRG for any fixed polynomial stretch $l(n) > n$.

▷ By Yao's theorem, it is equivalent to show that $\mathsf{BM}^{l(n)}(\cdot)$ is unpredictable (from the left).

- As before: We construct an algorithm $\mathcal{A}$ which tries to compute $\mathsf{hc}(x)$ given $f(x)$ using a given predictor $\mathcal{P}$ as a black-box subprocedure, and show that this implies that $\mathcal{P}$ can succeed only with negligible probability.

Wanted: $\mathcal{A}(1^n, f(x)) = h(x)$	Given: $\mathcal{P}(1^n, y_1 \dots y_{i-1}) = y_i$ for $y = \mathsf{BM}^l(x')$
$x \overset{u}{\in} \{0,1\}^n$	$i \overset{u}{\in} [l]$
$z := f(x)$	$x' \overset{u}{\in} \{0,1\}^n$
$\tilde{h} := \mathcal{A}(1^n, z)$	$y_1 \dots y_i := h(f^{l-1}(x')\|\| \dots \|\|h(f^{l-i+1}(x'))\|\|h(f^{l-i}(x'))$
	$\tilde{y}_i := \mathcal{P}(1^n, y_1 \dots y_{i-1})$
$\mathcal{A}$ wins iff $h(x) \overset{?}{=} \tilde{h}$	$\mathcal{P}$ wins iff $y_i = h(f^{l-i}(x')) \overset{?}{=} \tilde{y}_i$

1. Idea: use $\mathcal{P}$ to predict $h(x)$;

Wanted: $\mathcal{A}(1^n, f(x)) = h(x)$	Given: $\mathcal{P}(1^n, y_1 \dots y_{i-1}) = y_i$ for $y = \mathsf{BM}^l(x')$
$x \overset{u}{\in} \{0,1\}^n$	$i \overset{u}{\in} [l]$
$z := f(x)$	$x' \overset{u}{\in} \{0,1\}^n$
$\tilde{h} := \mathcal{A}(1^n, z)$	$y_1 \dots y_i := h(f^{l-1}(x')\|\| \dots \|\|h(f^{l-i+1}(x'))\|\|h(f^{l-i}(x'))$
	$\tilde{y}_i := \mathcal{P}(1^n, y_1 \dots y_{i-1})$
$\mathcal{A}$ wins iff $h(x) \overset{?}{=} \tilde{h}$	$\mathcal{P}$ wins iff $y_i = h(f^{l-i}(x')) \overset{?}{=} \tilde{y}_i$

1. Idea: use $\mathcal{P}$ to predict $h(x)$;
2. I.e. would like to have $y_i = h(f^{l-i}(x')) \overset{!}{=} h(x)$

Wanted: $\mathcal{A}(1^n, f(x)) = h(x)$	Given: $\mathcal{P}(1^n, y_1 \dots y_{i-1}) = y_i$ for $y = \mathsf{BM}^l(x')$
$x \overset{u}{\in} \{0,1\}^n$	$i \overset{u}{\in} [l]$
$z := f(x)$	$x' \overset{u}{\in} \{0,1\}^n$
$\tilde{h} := \mathcal{A}(1^n, z)$	$y_1 \dots y_i := h(f^{l-1}(x')\vert\vert \dots \vert\vert h(f^{l-i+1}(x'))\vert\vert h(f^{l-i}(x'))$
	$\tilde{y}_i := \mathcal{P}(1^n, y_1 \dots y_{i-1})$
$\mathcal{A}$ wins iff $h(x) \overset{?}{=} \tilde{h}$	$\mathcal{P}$ wins iff $y_i = h(f^{l-i}(x')) \overset{?}{=} \tilde{y}_i$

1. Idea: use $\mathcal{P}$ to predict $h(x)$;
2. I.e. would like to have $y_i = h(f^{l-i}(x')) \overset{!}{=} h(x)$
3. Sufficient: $f^{l-i}(x') = x$ resp. $x' = f^{i-l}(x)$ (Note: $i - l \leq 0$)

Wanted: $\mathcal{A}(1^n, f(x)) = h(x)$	Given: $\mathcal{P}(1^n, y_1 \dots y_{i-1}) = y_i$ for $y = \mathsf{BM}^l(x')$
$x \overset{u}{\in} \{0,1\}^n$	$i \overset{u}{\in} [l]$
$z := f(x)$	$x' \overset{u}{\in} \{0,1\}^n$
$\tilde{h} := \mathcal{A}(1^n, z)$	$y_1 \dots y_i := h(f^{l-1}(x')\|\| \dots \|\|h(f^{l-i+1}(x'))\|\|h(f^{l-i}(x'))$
	$\tilde{y}_i := \mathcal{P}(1^n, y_1 \dots y_{i-1})$
$\mathcal{A}$ wins iff $h(x) \overset{?}{=} \tilde{h}$	$\mathcal{P}$ wins iff $y_i = h(f^{l-i}(x')) \overset{?}{=} \tilde{y}_i$

1. Idea: use $\mathcal{P}$ to predict $h(x)$;
2. I.e. would like to have $y_i = h(f^{l-i}(x')) \overset{!}{=} h(x)$
3. Sufficient: $f^{l-i}(x') = x$ resp. $x' = f^{i-l}(x)$ (Note: $i - l \leq 0$)
4. Substitute $f^{i-l}(x)$ for x' on the right-hand side and simplify.

Wanted: $\mathcal{A}(1^n, f(x)) = h(x)$	Given: $\mathcal{P}(1^n, y_1 \ldots y_{i-1}) = y_i$ for $y = \mathsf{BM}^l(x')$
$x \overset{u}{\in} \{0,1\}^n$	$i \overset{u}{\in} [l]$
$z := f(x)$	$f^{i-l}(x) \overset{u}{\in} \{0,1\}^n$ (???)
$\tilde{h} := \mathcal{A}(1^n, z)$	$y_1 \ldots y_i := h(f^{l-1}(f^{i-l}(x)))\|\| \ldots \|\|h(f^{l-i}(f^{i-l}(x)))$
	$\tilde{y}_i := \mathcal{P}(1^n, y_1 \ldots y_{i-1})$
$\mathcal{A}$ wins iff $h(x) \overset{?}{=} \tilde{h}$	$\mathcal{P}$ wins iff $y_i = h(f^{l-i}(f^{i-l}(x))) \overset{?}{=} \tilde{y}_i$

1. Idea: use $\mathcal{P}$ to predict $h(x)$;
2. I.e. would like to have $y_i = h(f^{l-i}(x')) \overset{!}{=} h(x)$
3. Sufficient: $f^{l-i}(x') = x$ resp. $x' = f^{i-l}(x)$ (Note: $i - l \leq 0$)
4. Substitute $f^{i-l}(x)$ for x' on the right-hand side and simplify.

Wanted: $\mathcal{A}(1^n, f(x)) = h(x)$	Given: $\mathcal{P}(1^n, y_1 \ldots y_{i-1}) = y_i$ for $y = \mathsf{BM}^l(x')$
$x \overset{u}{\in} \{0,1\}^n$	$i \overset{u}{\in} [l]$
$z := f(x)$	$f^{i-l}(x) \overset{u}{\in} \{0,1\}^n$ (???)
$\tilde{h} := \mathcal{A}(1^n, z)$	$y_1 \ldots y_i := h(f^{i-1}(x))\|\| \ldots \|\|h(f(x))\|\|h(x)$
	$\tilde{y}_i := \mathcal{P}(1^n, y_1 \ldots y_{i-1})$
$\mathcal{A}$ wins iff $h(x) \overset{?}{=} \tilde{h}$	$\mathcal{P}$ wins iff $y_i = h(x) \overset{?}{=} \tilde{y}_i$

1. Idea: use $\mathcal{P}$ to predict $h(x)$;
2. I.e. would like to have $y_i = h(f^{l-i}(x')) \overset{!}{=} h(x)$
3. Sufficient: $f^{l-i}(x') = x$ resp. $x' = f^{i-l}(x)$ (Note: $i - l \leq 0$)
4. Substitute $f^{i-l}(x)$ for x' on the right-hand side and simplify.
5. Observe: $\mathcal{P}$ only needs $y_1 \ldots y_{i-1} = h(f^{i-1}(x))\|\| \ldots \|\|h(f(x))$ which $\mathcal{A}$ can compute directly as it is given $z = f(x)$.

Wanted: $\mathcal{A}(1^n, f(x)) = h(x)$	Given: $\mathcal{P}(1^n, y_1 \ldots y_{i-1}) = y_i$ for $y = \mathsf{BM}^l(x')$
$x \overset{u}{\in} \{0,1\}^n$	$i \overset{u}{\in} [l]$
$z := f(x)$	$f^{i-l}(x) \overset{u}{\in} \{0,1\}^n$ (???)
$\tilde{h} := \mathcal{A}(1^n, z)$	$y_1 \ldots y_i := h(f^{i-1}(x)) \| \ldots \| h(f(x)) \| h(x)$
	$\tilde{y}_i := \mathcal{P}(1^n, y_1 \ldots y_{i-1})$
$\mathcal{A}$ wins iff $h(x) \overset{?}{=} \tilde{h}$	$\mathcal{P}$ wins iff $y_i = h(x) \overset{?}{=} \tilde{y}_i$

1. Idea: use $\mathcal{P}$ to predict $h(x)$;
2. I.e. would like to have $y_i = h(f^{l-i}(x')) \overset{!}{=} h(x)$
3. Sufficient: $f^{l-i}(x') = x$ resp. $x' = f^{i-l}(x)$ (Note: $i - l \leq 0$)
4. Substitute $f^{i-l}(x)$ for x' on the right-hand side and simplify.
5. Observe: $\mathcal{P}$ only needs $y_1 \ldots y_{i-1} = h(f^{i-1}(x)) \| \ldots \| h(f(x))$ which $\mathcal{A}$ can compute directly as it is given $z = f(x)$.
6. Finally: as f is a bijection, it does not matter if we choose $x \overset{u}{\in} \{0,1\}^n$ or $f^{i-1}(x) \overset{u}{\in} \{0,1\}$ or $f(x) = z \overset{u}{\in} \{0,1\}^n$.

Proof of the Blum-Micali construction

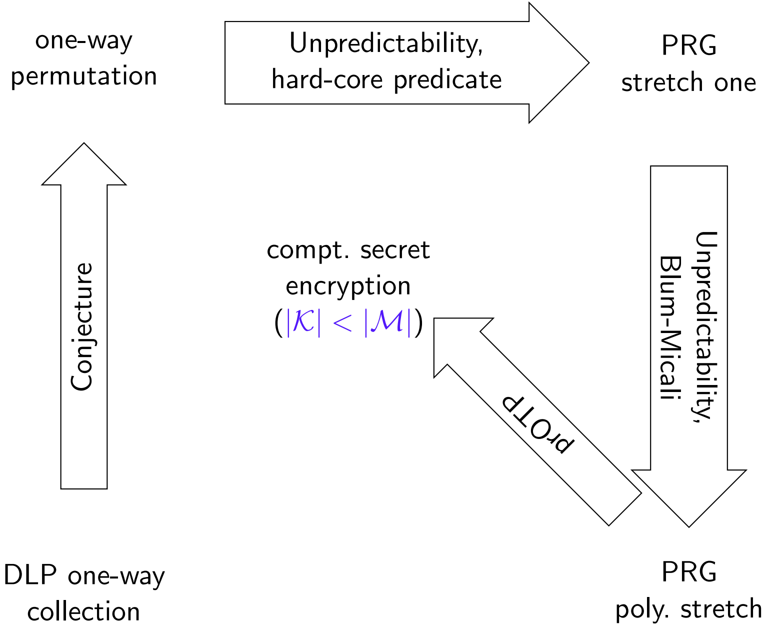
Alice&Bob	$\mathcal{A}(1^n, f(x)) = h(x)$ using $\mathcal{P}$
$x \overset{u}{\in} \{0,1\}^n$	$i \overset{u}{\in} [l]$
$z := f(x)$	$y_1 \dots y_{i-1} := h(f^{i-2}(z))\|\| \dots \|\|h(z))$
$\tilde{h} := \mathcal{A}(1^n, z)$	return $\tilde{h} := \mathcal{P}(1^n, y_1 \dots y_{i-1})$
$\mathcal{A}$ wins iff $h(x) \overset{?}{=} \tilde{h}$	

1. Idea: use $\mathcal{P}$ to predict $h(x)$;
2. I.e. would like to have $y_i = h(f^{l-i}(x')) \overset{!}{=} h(x)$
3. Sufficient: $f^{l-i}(x') = x$ resp. $x' = f^{i-l}(x)$ (Note: $i - l \leq 0$)
4. Substitute $f^{i-l}(x)$ for x' on the right-hand side and simplify.
5. Observe: $\mathcal{P}$ only needs $y_1 \dots y_{i-1} = h(f^{i-1}(x))\|\| \dots \|\|h(f(x))$ which $\mathcal{A}$ can compute directly as it is given $z = f(x)$.
6. Finally: as f is a bijection, it does not matter if we choose $x \overset{u}{\in} \{0,1\}^n$ or $f^{i-1}(x) \overset{u}{\in} \{0,1\}$ or $f(x) = z \overset{u}{\in} \{0,1\}^n$.
7. Yields above definition for $\mathcal{A}$.

- Note that the proof requires that $\mathsf{hc}(x)$ is the right-most/last bit in the output of BM.

  The reason for this is simply the definition of unpredictability which goes from left to right.

- **Ex**: Show that $G_l(x) := f^l(x)||\mathsf{BM}^l(x)$ is a PRG of fixed stretch for every fixed l polynomial in n.
  - Discuss the advantages/disadvantages of outputting also $f^l(x)$.
  - In particular, consider the case when a TDP is used for f and the resulting PRG is used within the prOTP.


one-way
permutation
Unpredictability,
hard-core predicate
PRG
stretch one
Conjecture
compt. secret
encryption
($|\mathcal{K}| < |\mathcal{M}|$)
prOTP
Unpredictability,
Blum-Micali
DLP one-way
collection
PRG
poly. stretch

1. Lecture 16 - 18 – OWF candidates, Construction of PRGs, HF

- Informally: (only interface)

  Any easy to compute function from "$\{0,1\}^*$" to $\{0,1\}^{l_{\text{out}}}$

  - Easy to compute: e.g. linear in the input length.
  - Read "$\{0,1\}^*$" as "practically unbounded input",

    e.g. all inputs up to 2^{1024} bits ($= 2^{996}$ TB).

- **Definition**:

  Let $l_{\text{out}} \in \mathbb{N}$ and $h\colon \mathcal{M} \to \{0,1\}^{l_{\text{out}}}$ a DPT-computable function.

  h is a compression function if $\mathcal{M} = \{0,1\}^{l_{\text{in}}}$ for some $l_{\text{in}} > l_{\text{out}}$.

  h is a hash function if $\mathcal{M} = \{0,1\}^{<2^l}$ for some $l > 0$.

▷ Ideally, computation of $h(m)$ takes $\mathcal{O}(|m|)$ time.

- In the design of efficient algorithms and data structures:
  - Given: Universe U of possible data, hash values $\{0,1\}^{l_{\text{out}}}$.
  - Goal: Find a hash function $h\colon U \to \{0,1\}^{l_{\text{out}}}$ such that for any unknown selection $S \subseteq U$ the number of collisions is "small".
    - Collision: Any two x, x' such that $h(x) = h(x')$.
  - Ideally: "h distributes S uniformly over the hash values"

    $\forall y \in \{0,1\}^{l_{\text{out}}}\colon \left|h^{-1}(y) \cap S\right| \approx |S| / 2^{l_{\text{out}}}$
    - In general not possible; for a fixed h simply choose $S \subseteq h^{-1}(y)$ for some suitable $y \in \{0,1\}^{l_{\text{out}}}$.
    - Solution: randomly choose h from a parametrized familiy, see universal hashing.

- For cryptographic uses, a hash function $h\colon \mathcal{M} \to \{0,1\}^{l_{\text{out}}}$ has to satisfy further requirements, e.g. (informally):

  - Collision resistance:

    It is computationally infeasible to find m_1, m_2 s.t. $h(m_1) = h(m_2)$ and $m_1 \neq m_2$.

  - Second-preimage resistance:

    Given m_1, it is computationally infeasible to find m_2 with $h(m_1) = h(m_2)$ and $m_1 \neq m_2$.

  - Preimage resistance:

    Given $h(m_1)$, it is computationally infeasible to find any m_2 with $h(m_1) = h(m_2)$. ("one-wayness")

- ▷ What means "computationally infeasible"?
  - ▷ Need to fix either asymptotic or concrete bounds on the resources and success probability of the adversary.
- ▷ Preimage and second-preimage resistance: How is m_1 chosen?
  - Second-preimage resistance is meaningless if we are allowed to choose m_1 deterministically.
  - ▷ Either let the advesary choose m_1 or $m_1 \overset{u}{\in} \{0,1\}^{L(n)} \subseteq \mathcal{M}$.
- ▷ Any function $h\colon A \to B$ with $|A| > |B|$ has always a collision, i.e. a pair $m_1 \neq m_2$ with $h(m_1) = h(m_1)$.
  - There always exists an efficient adversary which simply outputs m_1, m_2 for a function.
  - ▷ Thus, consider collections (families) of hash functions.

- **Definition**:

  Let $\mathcal{H} = (\mathsf{Gen}, h)$ be a function collection such that

  $h_I \colon \mathcal{M}_n \to \{0,1\}^{l_{\mathsf{out}}(n)}$ for any I output by $\mathsf{Gen}(1^n)$ where $l_{\mathsf{out}}(n)$ is a polynomial.

▷ $\mathcal{H}$ is a collection of compression functions (CCF) if

  $\mathcal{M}_n = \{0,1\}^{l_{\mathsf{in}}(n)}$ for some polynomial $l_{\mathsf{in}}(n) > l_{\mathsf{out}}(n)$.

▷ $\mathcal{H}$ is a collection of hash functions (CHF) if

  $\mathcal{M} = \{0,1\}^{<2^{l(n)}}$ for some polynomial $l(n) > 0$.

- **Example**: DLP-CCF

  Gen: on input 1^n, run $\mathsf{Gen}\mathbb{QR}_{\mathsf{safe}}(1^n)$ to obtain (p, q, g), then choose $x \overset{u}{\in} \mathbb{Z}_q$, and set $r := g^x \bmod p$. Output $I = (p, q, g, r)$.

  h: on input $I = (p, q, g, r)$ and $(u, v) \in \mathbb{Z}_q \times \mathbb{Z}_q$ output $h_{(p,q,g,r)}(u, v) := g^u \cdot r^v \bmod p$.

- Assume that $2^n \leq q \leq p \leq 2q + 1 \leq 2^{n+1} - 1$.

▷ Then any n-bit string u represents some element in $\mathbb{Z}_q$, and

any $x \in \mathbb{Z}_p^* \subseteq \mathbb{Z}_p$ can be represented as a $n + 1$-bit string.

▷ I.e. h_I compresses $l_{\mathsf{in}}(n) = 2n$-bit strings to $l_{\mathsf{out}}(n) = n + 1$-bit strings.

- [11] discusses several formalizations of the preceding informal requirements, and studies their relation.
- In total, seven formalizations are obtained from the informal requirements depending on
  - ▷ whether the adversary chooses m_1, or if m_1 is chosen uniformly at random from a finite subset of $m_1 \overset{u}{\in} \{0,1\}^{L(n)} \subseteq \mathcal{M}_n$. ($L(n) = l_{\mathsf{in}}(n)$ for a collection of compression functions.)
  - ▷ whether the adversary may pick a function from the collection, or if the function is generated randomly by Gen.
- We consider only the four most important definitions where the concrete hash function h_I is generated via $I := \mathsf{Gen}(1^n)$.

- **Definition**: Let $\mathcal{H} = (\mathsf{Gen}, h)$ be a CCF or CHF, and $L(n)$ any polynomial such that $\{0,1\}^{L(n)} \subseteq \mathcal{M}_n$.

Game $\textsc{Coll}$	Game $\mathrm{UOWHF}[L(n)]$	Game $\textsc{Sec}[L(n)]$	Game $\textsc{Pre}[L(n)]$
	$m_1 \stackrel{r}{:=} \mathcal{A}(1^n) \in \{0,1\}^{L(n)}$	$m_1 \stackrel{u}{\in} \{0,1\}^{L(n)}$	$m_1 \stackrel{u}{\in} \{0,1\}^{L(n)}$
$I \stackrel{r}{:=} \mathsf{Gen}(1^n)$	$I \stackrel{r}{:=} \mathsf{Gen}(1^n)$	$I \stackrel{r}{:=} \mathsf{Gen}(1^n)$	$I \stackrel{r}{:=} \mathsf{Gen}(1^n)$
$(m_1, m_2) \stackrel{r}{:=} \mathcal{A}(I)$	$m_2 \stackrel{r}{:=} \mathcal{A}(I, m_1)$	$m_2 \stackrel{r}{:=} \mathcal{A}(I, m_1)$	$m_2 \stackrel{r}{:=} \mathcal{A}(I, h_I(m_1))$
$\mathsf{Win}_{n,\mathcal{H}}^{\textsc{Coll}}(\mathcal{A})$:	$\mathsf{Win}_{n,\mathcal{H}}^{\mathrm{UOWHF}[L(n)]}(\mathcal{A})$:	$\mathsf{Win}_{n,\mathcal{H}}^{\textsc{Sec}[L(n)]}(\mathcal{A})$:	$\mathsf{Win}_{n,\mathcal{H}}^{\textsc{Pre}[L(n)]}(\mathcal{A})$:
$h_I(m_1) = h_I(m_2)$	$h_I(m_1) = h_I(m_2)$	$h_I(m_1) = h_I(m_2)$	$h_I(m_1) = h_I(m_2)$
and $m_1 \neq m_2$	and $m_1 \neq m_2$	and $m_1 \neq m_2$	

If the respective winning probability is negligigble w.r.t. n for any admissible PPT-adversary $\mathcal{A}$, then $\mathcal{H}$ is

- $\textsc{Coll}$: collision resistant
- $\mathrm{UOWHF}[L(n)]$: a universal one-way hash function w.r.t. inputs of length $L(n)$
- $\textsc{Sec}[L(n)]$: second-preimage resistant w.r.t. inputs of length $L(n)$
- $\textsc{Pre}[L(n)]$: preimage resistant (one-way) w.r.t. inputs of length $L(n)$

- **Theorem** [11]:

  Let $\mathcal{H} = (\mathsf{Gen}, h)$ be a CCF/CHF of output length $l_{\text{out}}(n)$.

  If $\mathcal{H}$ is collision resistant, then it is a UOWHF for any $L(n)$.

  If $\mathcal{H}$ is a UOWHF for $L(n)$, then it is second-pre. resistant for $L(n)$.

  If $\mathcal{H}$ is second-preimage resistant for $L(n)$, and $2^{l_{\text{out}}(n)-L(n)}$ is negligible, then it is also preimage resistant for $L(n)$.

  ▷ The term $2^{l_{\text{out}}(n)-L(n)}$ is esssentially the prob. that $h_I(m_1)$ has a unique preimage within $\{0,1\}^{L(n)}$.

  - **Ex**: Assume that (Gen, g) is collision resistant with output length $l_{\text{out}}(n)$. Let $h_I(x) := 1x$ if $|x| = l_{\text{out}}(n)$; otherwise $h_I(x) := 0g_I(x)$.

    Show that (Gen, h) is also collision resistant, but not preimage resistant for inputs of length $L(n) = l_{\text{out}}(n)$.

- **Theorem**: UOWHFs can be constructed from OWFs. [12]

- **Conjecture**: OWFs not enough for collision resistance. [14]

- **Lemma**:

  Assume that the DLP is hard relative to $\mathsf{Gen}\mathbb{QR}_{\mathsf{safe}}$.

  Then the DLP-CCF is collision resistant.

▷ Proof: Let $\mathcal{A}$ be a PPT-collision attack on (Gen, h).

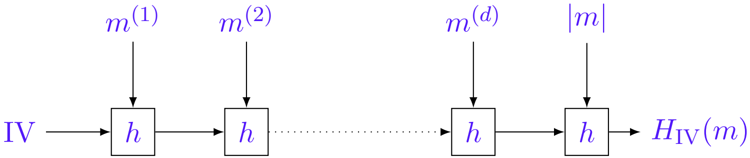
  Define $\mathcal{B}$ as follows:

  - Input: (p, q, g) and $r = g^x \bmod p$ for some secret $x \overset{u}{\in} \mathbb{Z}_q$.
  - If $r = 1$, output $x = 0$.
  - Otherwise, pass (p, q, g, r) to $\mathcal{A}$ to obtain $(a, b) \neq (u, v)$.
  - If $h_I(a, b) \neq h_{I,r}(u, v)$, output any element in $\mathbb{Z}_q$.
  - Otherwise return $(a - u) \cdot (v - b)^{-1} \mod q$.

  Ex: Determine the prob. that $\mathcal{B}$ succeeds in computing a logarithm of r modulo p. (Why is it important that q is prime?)

- In practice, many hash functions are constructed from compression functions by means of the Merkle-Damgård construction. [9]

▷ E.g.: SHA-1, SHA-2, RIPEMD, Grøstl



- This construction was also used in NMAC and HMAC.
- Recall: For NMAC the compression functions are PRFs.
  - ▷ Many modern hash functions (in particular the SHA-3 candidates) have been designed with that in mind.
  - ▷ Grøstel's compression function is based on AES.

- **Definition**: Let $h : \{0,1\}^{l_{\text{in}}} \to \{0,1\}^{l_{\text{out}}}$ be a compression function with $\delta := l_{\text{in}} - l_{\text{out}} > 0$.

  Let $\text{pad}_{\text{MD}}(m) := m||0^p||\lfloor|m|\rceil$ such that $|\text{pad}_{\text{MD}}(m)|$ is a minimal multiple of δ and $\lfloor|m|\rceil$ is encoded using exactly δ bits.[1]

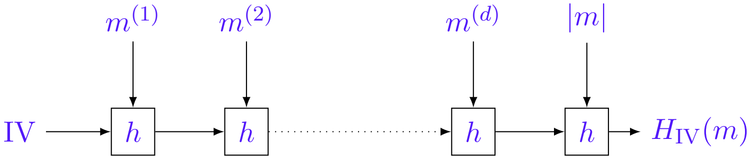
  For any $\text{IV} \in \{0,1\}^{l_{\text{out}}}$, and $m \in \{0,1\}^*$ with $|m| < 2^\delta$, define $H_{\text{IV}}(x) := z^{(t)}$ where:

  $z^{(0)} := \text{IV}$ and $z^{(i)} := h(z^{(i-1)}||m^{(i)})$ for $\text{pad}_{\text{MD}}(m) = m^{(1)}||\ldots||m^{(t)}$.

  For a CCF (Gen, h) apply the construction to each h_I yielding $H_{I,\text{IV}}$.

- **Ex**: Adapt the DLP-CCF so that the Merkle-Damgård construction can be applied to it.

[1] This is not the only possible choice, but it suffices for us.



- **Theorem**: [9]

  Let (Gen, h) be a collision-resistant CCF.

  Construct (Gen, H) from (Gen, h) using the Merkle-Damgård construction.

  Then (Gen, H) is a collision-resistant CHF for any fixed IV.

- The IV can be treated as a further function parameter of the CHF.

  It is only important, that the IV is fixed by the function parameters I so that Alice and Bob uses the same IV.

  Recall for F-NMAC we indeed need to be able to change the IV.

- Fix any IV. We show that any collision of $H_I := H_{I,\mathrm{IV}}$ yields a collision of h_I.

  To this end, assume that $H_I(x) = H_I(y)$ for some $x \neq y \in \{0,1\}^*$.

- Let $\mathsf{pad}_{\mathsf{MD}}(x) = x^{(1)} \dots x^{(d)} x^{(d+1)}$ resp. $\mathsf{pad}_{\mathsf{MD}}(y) = y^{(1)} \dots y^{(t)} y^{(t+1)}$.

- Let $u^{(i)}$ be the intermediate values obtained from x, i.e.

  $u^{(0)} = \mathrm{IV}$ and $u^{(i)} = h_I(u^{(i-1)} || x^{(i)})$

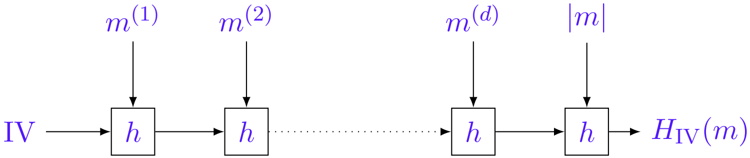
  Analogously for $v^{(i)}$ and $y^{(i)}$.

- Recall that by definition, $|x|, |y|$ fit into a single block, i.e.

  $x^{(d+1)} = y^{(t+1)} \Rightarrow |x| = |y|$ s.t. $d = t$.

- Assume first $x^{(d+1)} \neq y^{(t+1)}$.

  Then: $u^{(d)} || x^{(d+1)} \neq v^{(t)} || y^{(t+1)}$ is a collision of h_I.



- Assume thus $x^{(d+1)} = y^{(t+1)}$, i.e. $t = d$ and $|x| = |y|$.

  As $x \neq y$, there is some $i \in [d+1]$ s.t. $x^{(i)} \neq y^{(i)}$.

  Hence, there is also some maximal index m s.t. $u^{(m-1)}||x^{(m)} \neq v^{(m-1)}||y^{(m)}$.

▷ If $m = d+1$, then $u^{(d)}||x^{(d+1)} \neq v^{(d)}||y^{(d+1)}$ and

$h_I(u^{(d)}||x^{(d+1)}) = H_I(x) = H_I(y) = h_I(v^{(d)}||x^{(d+1)})$.

▷ If $m \leq d$ and m is maximal, we need to have $u^{(m)} = v^{(m)}$.

Thus, $h_I(u^{(m-1)}||x^{(m)}) = u^{(m)} = v^{(m)} = h_I(v^{(m-1)}||y^{(m)})$.

- Similar to block ciphers, the output length l_{out} of a hash function needs to be large enough so that the prob. of a collision is negligible:

▷ Assume we have a compression function $h\colon \{0,1\}^{2l} \to \{0,1\}^l$.

Let $X_1, \ldots, X_q$ be independent RV uniformly distributed on $\{0,1\}^{2l}$.

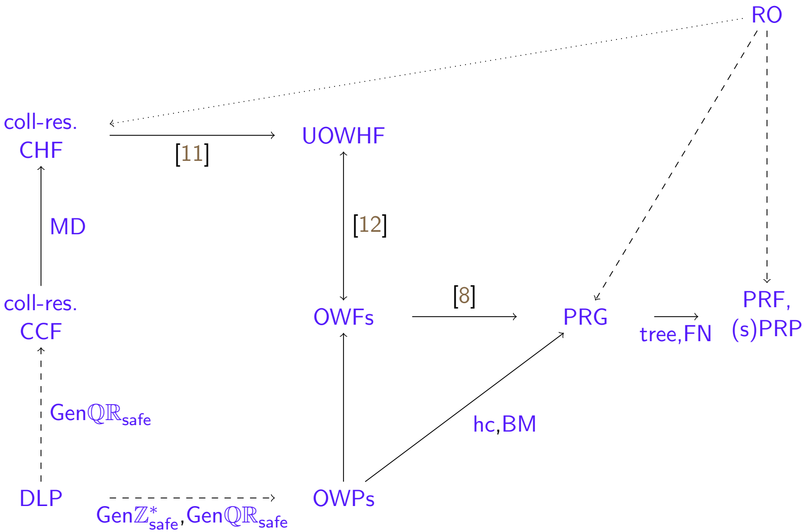
Intuitively, the best that h can do is to map $\frac{2^{2l}}{2^l}$ inputs on the same output, i.e. $\left|h^{-1}(y)\right| = 2^l$ for every y.

- If for some y the set of preimages is significantly larger than 2^l, the prob. of a collision only increases. [3]

Then, $h(X_i)$ is uniformly distributed on $\{0,1\}^l$. Probability of a collision:

- $\Theta(\binom{q}{2}2^{-l})$ within $\{h(X_1), \ldots, h(X_q)\}$.
- $\Theta(\binom{q}{2}2^{-2l})$ within $\{X_1, \ldots, X_q\}$ (exponentially smaller).

That is, a collision $h(X_i) = h(X_j)$ results almost always from $X_i \neq X_j$.


RO
coll-res. CHF
[11]
UOWHF
MD
[12]
coll-res. CCF
OWFs
[8]
PRG
tree,FN
PRF, (s)PRP
Gen$\mathbb{QR}_{\text{safe}}$
hc,BM
DLP
Gen$\mathbb{Z}^*_{\text{safe}}$,Gen$\mathbb{QR}_{\text{safe}}$
OWPs

[1] D. Aggarwal and U. Maurer. *Breaking RSA Generically is Equivalent to Factoring*. URL: http://eprint.iacr.org/2008/260.pdf.

[2] S. Arora and B. Barak. *Computational complexity: a mordern approach*. URL: http://www.cs.princeton.edu/theory/index.php/Compbook/Draft.

[3] M. Bellare and T. Kohno. *Hash Function Balance and its Impact on Birthday Attacks*. URL: http://www.cs.washington.edu/homes/yoshi/papers/Hash/balance.pdf.

[4] M. Blum and S. Micali. *How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits*. URL: http://www.csee.wvu.edu/~xinl/library/papers/comp/Blum_FOCS1982.pdf.

[5] J. Buchmann. *Einführung in die Kryptographie*.

[6] N. Ferguson and B. Schneier. *Practical cryptography*.

[7] O. Goldreich and L. Levin. *A Hard-Core Predicate for all One-Way Functions*. URL: http://www.wisdom.weizmann.ac.il/~oded/X/gl.pdf.

[8] J. Håstad et al. *Construction of a Pseudo-Random Generator From Any One-Way Function*. URL: http://www.cs.umd.edu/~gasarch/oneway/HILL.pdf.

[9] R. Merkle. *PhD thesis*. URL: http://www.merkle.com/papers/Thesis1979.pdf.

[10] M. Näslund. *Bit Extraction, Hard-Core Predicates, and the Bit Security of RSA*. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.6.3112&rep=rep1&type=pdf.

[11] P. Rogaway and T. Shrimpton. *Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance*. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.62.539&rep=rep1&type=pdf.

[12] J. Rompel. *One-way functions are necessary and sufficient for secure signatures*. URL: http://www.cs.umd.edu/~jkatz/papers/rompel.pdf.

[13] V. Shoup. *Lower Bounds for Discrete Logarithms and Related Problems*. URL: http://www.shoup.net/papers/dlbounds1.pdf.

[14] D. Simon. *Finding Collisions on a One-Way Street: Can Secure Hash Functions be Based on General Assumptions*. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.136.4654.

[15] A. Yao. *Theory and Applications of Trapdoor Functions (Extended Abstract)*. URL: http://www.busim.ee.boun.edu.tr/~mihcak/teaching/ee684-spring07/proposed-project-papers/one-way-functions/Yao-XOR-Lemma-and-Hard-Core-Predicates/Yao-XOR-original.pdf.