Pseudo-Random Functions, Factoring and Assumptions

Moni Naor

Omer Reingold

Alon Rosen

Weizmann Institute

WEIZMANN INSTITUTE OF SCIENCE

Rigorous Specification of Security

To define security of a system must specify:

1. The power of the **adversary** - **both computational and access to the system.**

2. **What constitute a failure of the system - what it means to break the system.**

Hardness assumption: no machine with limited resources can solve instances of a problem. More specifically:

- Distribution on instances,
- Prob. of success.

A security proof of a system *implementation*: no adversary with limitation as (1) can break the system as in (2), assuming the hardness assumption.

Cryptographic Assumptions

- Almost any Cryptographic Tasks has to be based on computational hardness
- State of the art of Complexity Theory
  $\Rightarrow$ must make assumptions about computational hardness of problems.

How to choose a reasonable assumption?

One natural approach:

- Use a well studied problem
- The complexity of proposed breaking algorithms should be well understood

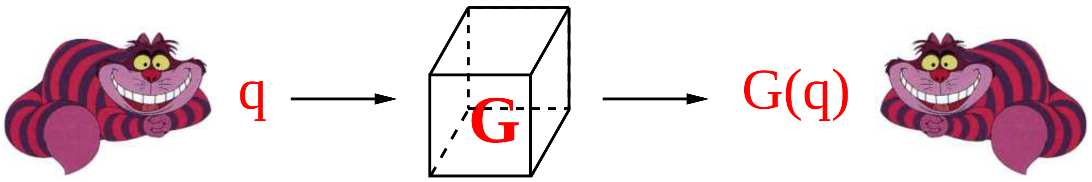
The best established problem - factoring integers

Pseudo-Random Functions

Distribution $\{f_i : \{0,1\}^n \mapsto \{0,1\}^n\}_i$ which is

- **Efficient:** i is short and $\forall i$, easy to compute f_i.
- **Pseudo-Random:** f_k cannot be efficiently distinguished **from a random function.**

The **distinguisher's** access to the function:
adaptive black-box attack

(1) 

(2) Is G random ? Does $G = f_k$?

Applications:

- Private-key encryption.
- Authentication.
- Identification.
- Complexity.
- Learning.
- More...

Previous constructions

- **The GGM construction** [GGM84]: Generic, based on pseudo-random generators, requires n^2 modular multiplications (when based on factoring).

- **The NR construction** [NR95]: Generic, based on pseudo-random synthesizers:

  - **DDH construction** [NR97]: Specific, based on DDH, requires $2n$ modular multiplications.

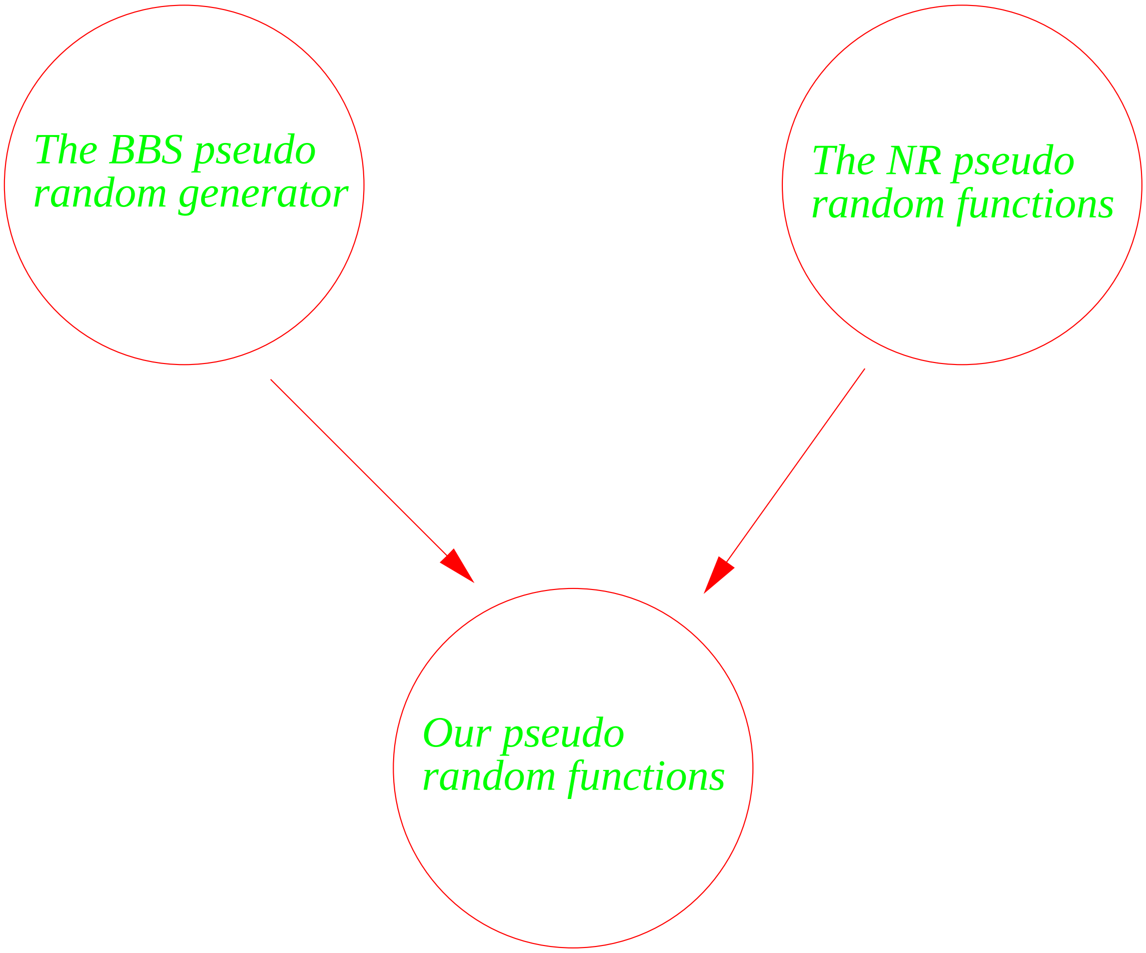
  - **Factoring construction** [NR97]: Specific, based on factoring, requires $2n$ modular multiplications, single bit as output.

We focus on **efficiency** (number of modular mult).

Our construction

- **Factoring based**, requires $3n$ modular multiplications, n bits as output.

Our Construction



The BBS generator plays multiple roles in the construction.

The Factoring Assumption

Preliminaries:

- $N = P \cdot Q$: P and Q distinct primes of large size.
- $[N]$: $\{0, 1, \ldots, N-1\}$.
- n: size of N (in bits).
- Z_N^*: multiplicative group $\bmod N$
  (of all $x \in [N]$ such that $\gcd(x, N) = 1$).
- $\varphi(N)$: size of Z_N^* (equals $(P-1)(Q-1)$).
- All exponentiations/multiplications are $\bmod N$.

Assumption (factoring):
For any probabilistic poly-time algorithm $\mathcal{A}$, for every positive constant α:

$$\Pr\left[\mathcal{A}(P \cdot Q) \in \{P, Q\}\right] < \frac{1}{n^{\alpha}}$$

The BM Pseudo-Random Generator

Generic construction:

- f: One-way Permutation ($\{0,1\}^n \rightarrow \{0,1\}^n$).
- $\mathcal{B}$: Hard-core predicate for f ($\{0,1\}^n \rightarrow \{0,1\}$).

The function $G_\ell^{BM} : \{0,1\}^n \rightarrow \{0,1\}^\ell$ defined as:

$$G_\ell^{BM}(x) = \mathcal{B}(x), \mathcal{B}(f(x)), \ldots, \mathcal{B}(f^{(k)}(x)), \ldots, \mathcal{B}(f^{(\ell(n)-1)}(x))$$

is a pseudo-random generator.

The BBS Pseudo-Random Generator

The BM construction based on factoring

Candidate One-way Permutation:

- **Squaring:** $x \mapsto x^2 \bmod N$ is one way.
  (assuming factoring is hard [Rabin79])

- **Problem:** not necessarily a permutation.

- **Solution:**

  – Restrict N to be a Blum-integer.

  – Squaring is now a permutation over Q_N.

Blum-integer: $N = P \cdot Q$ where P and Q are large primes and $P = Q = 3 \bmod 4$.

Quadratic-residues(Q_N):Subgroup of elements in Z_N^* which have a square root.

BBS cont.

A Hard-core Predicate:

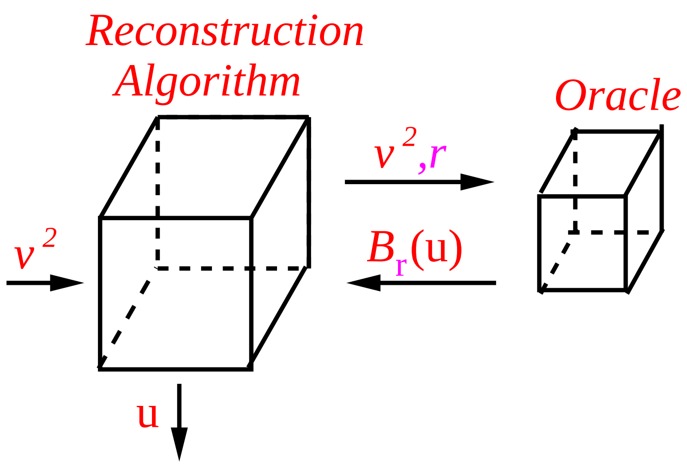
Goldreich-Levin:
A Hard-core predicate for "all" one-way functions:

$$\mathcal{B}_r(m) = \langle m, r \rangle$$

($\langle m, r \rangle$ inner product mod2).

Example:

- $v \in Z_N^*$, $u \in Q_N$ such that $u^2 = v^2 \bmod N$.
- Then $\langle v^2, r \rangle \mapsto \mathcal{B}_r(u)$ hard.



BBS cont.

- One-way Permutation: $x \mapsto x^2 \bmod N$.

- Hard-core predicate: $\mathcal{B}_r(m) = \langle m, r \rangle$.
  ($\langle m, r \rangle$ inner product mod2).

The function $G^{BBS}_{N,r,\ell} : \{0,1\}^n \to \{0,1\}^\ell$ defined as:

$$G^{BBS}_{N,r,\ell}(x) = \mathcal{B}_r(x), \mathcal{B}_r(x^2), \ldots, \mathcal{B}_r(x^{2^k}), \ldots, \mathcal{B}_r(x^{2^{\ell(n)-1}})$$

is a pseudo-random generator.

- Efficient: 1 modular multiplication per bit
  (in particular, requires $2n$ multiplications to double the length of an n-bit seed).

- At least as secure as factoring Blum-integers.

Remark 1 Original construction used $\mathcal{LSB}$.

The Easy-Access Problem

- G^{BBS} defines an infinite sequence of bits:

$$\mathcal{B}_r(x), \mathcal{B}_r(x^2), \ldots, \mathcal{B}_r(x^{2^k}), \ldots$$

Given $\varphi(N)$:

- If $i < 2^{\mathrm{poly}(n)}$, then i^{th} bit can be computed in poly(n) time:

1. Compute $\beta_i = 2^{i-1} \bmod \varphi(N)$
   (repeated squaring, $O(\log(i))$ multiplications).
2. Compute $b_i = \mathcal{B}_r(x^{2^{i-1}}) = \mathcal{B}_r(x^{\beta_i})$.

Question: Is the function $f : \{0,1\}^n \to \{0,1\}$

$$f_{N,g,r}^{BBS1}(i) \stackrel{\text{def}}{=} \mathcal{B}_r(g^{2^{i-1}})$$

a pseudo-random function?

From Generators to Functions

The GGM construction [GGM84]:

- Motivated by the easy-access problem.
- Sequential, requires n invocations of a length doubling pseudo-random generator.
- When applied to G^{BBS}, yields a factoring-based pseudo-random function which requires as much as n^2 modular multiplications.

However

If f^{BBS1} would have been pseudo-random:

The function $f : \{0,1\}^n \to \{0,1\}^n$ defined as:

$$f^{BBSn}_{N,g,r}(i) \stackrel{\text{def}}{=} G^{BBS}_{N,r,n}(g^{2^{i \cdot n}}) =$$

$$\mathcal{B}_r(g^{2^{in}}), \mathcal{B}_r(g^{2^{in+1}}), \ldots, \mathcal{B}_r(g^{2^{(i+1)n-1}})$$

is pseudo-random and is more efficient than GGM (requires only $3n$ modular multiplications).

The NR Pseudo-Random Functions

A function which matches the efficiency of f^{BBS1}.

Not known to be pseudo random

A key of the function is $\langle N, \vec{a}, g, r \rangle$.

- N: Blum-Integer.
- $\vec{a} = \langle {}^{a_{1,0},}_{a_{1,1},} \; {}^{a_{2,0},}_{a_{2,1},} \; \ldots \; {}^{,a_{n,0}}_{,a_{n,1}} \rangle$
  Uniformly distributed seq. of $2n$ elements in $[N]$.
- g: Uniformly distributed quadratic residue in Z_N^*.
- r: Uniformly distributed n-bit string.

For any n-bit input, $x = x_1 x_2 \cdots x_n$, the function $f_{N,\vec{a},g,r}$ (with 1 bit of output) is defined to be:

$$f_{N,\vec{a},g,r}(x) \stackrel{\text{def}}{=} \mathcal{B}_r(g^{\prod_{i=1}^{n} a_{i,x_i}})$$

NR Functions cont.

$$f_{N,\vec{a},g,r}(x) = \mathcal{B}_r(g^{\prod_{i=1}^n a_{i,x_i}})$$

The value of x determines a_{i,x_i}'s to be multiplied:
Example:
$x = 100\ldots1$

$$\left\langle \begin{matrix} a_{1,0} & ,a_{2,x_2} & ,a_{3,x_3}, & & ,a_{n,0} \\ a_{1,x_1} & ,a_{2,1} & ,a_{3,1} & , \ldots & ,a_{n,x_n} \end{matrix} \right\rangle$$

Features:

- Efficient: $2n$ multiplications per evaluation.
    1. Compute $y = \prod_{i=1}^n a_{i,x_i}$ mod $\varphi(N)$.
    2. Compute g^y mod N.
- At least as secure as factoring Blum-integers.
- Matches the efficiency of f^{BBS1}.

The new idea:

Use $g^{\prod_{i=1}^n a_{i,y_i}}$ as a seed to the BBS generator.

Our Pseudo-Random Functions

A key of the function is $\langle N, \vec{a}, g, r \rangle$ (as before).

For any n-bit input, $x = x_1 x_2 \cdots x_n$, the function $f_{N,\vec{a},g,r}$ (with ℓ bits of output) is defined to be:

$$f_{N,\vec{a},g,r}(x) \stackrel{\text{def}}{=} G^{BBS}_{N,r,\ell}(g^{\prod_{i=1}^{n} a_{i,x_i}}) =$$

$$\mathcal{B}_r(g^{\prod_{i=1}^{n} a_{i,x_i}}), \mathcal{B}_r(g^{2\prod_{i=1}^{n} a_{i,x_i}}), \ldots, \mathcal{B}_r(g^{2^k \prod_{i=1}^{n} a_{i,x_i}}), \ldots, \mathcal{B}_r(g^{2^{\ell-1} \prod_{i=1}^{n} a_{i,x_i}})$$

Features:

- Efficient: $3n$ multiplications per evaluation. (take $\ell(n) = n$)
  1. Compute $y = \prod_{i=1}^{n} a_{i,x_i}$ mod $\varphi(N)$.
  2. Compute $z = g^y$ mod N.
  3. Compute $\mathcal{B}_r(z), \mathcal{B}_r(z^2), \ldots, \mathcal{B}_r(z^{2^{n-1}})$.
- At least as secure as factoring Blum-integers.

What is going on

Consider the function $h_{N,\vec{a},g}(x) = g^{\prod_{i=1}^{n} a_{i,x_i}}$

- Unpredictable under an adaptive sample and a random challenge [NR98].

Suggested methodology:

1. Take an "unpredictable" function $h_s : \{0,1\}^n \rightarrow \{0,1\}^n$.
2. Take a pseudo-random generator $G : \{0,1\}^n \rightarrow \{0,1\}^\ell$.
3. Obtain a pseudo-random function $f : \{0,1\}^n \rightarrow \{0,1\}^\ell$ by setting $f_s(x) = G(h_s(x))$.

– Works when h_s is pseudo-random.

– Does not necessarily work when h_s is unpredictable.

Nevertheless, our functions are pseudo-random.

A Counter Example

1.(a) Let $h'_s : \{0,1\}^n \to \{0,1\}^{\frac{n}{2}}$ be unpredictable.

(b) Define $h_s : \{0,1\}^n \to \{0,1\}^n$ as $h_s(x) = \langle h'_s(x), \vec{0} \rangle$.

(c) h_s is also an unpredictable function.

2.(a) Let $G' : \{0,1\}^{\frac{n}{2}} \to \{0,1\}^{\ell}$ be pseudo-random.

(b) For $z, y \in \{0,1\}^{\frac{n}{2}}$ define $G : \{0,1\}^n \to \{0,1\}^{\ell}$ as $G(z,y) = G'(y)$.

(c) G is also a pseudo-random generator.

However:
The function $f : \{0,1\}^n \to \{0,1\}^{\ell}$ defined as:
$f_s(x) = G(h_s(x))$ is always equal to $G'(\vec{0})$.

Obviously, f_s cannot be pseudo-random.

Proof Ideas

- Use hybrid argument (everyone does). Use it in reverse. Analogous to the principle of deferred decision.

- Use of GL-hardcore bit is crucial. Need to fix x and vary r. $\mathcal{LSB}$ predicate does not work.

- Randomize the input as BBR

Proof Outline

Assume there exists a PPTM $\mathcal{M}$ which distinguishes $f_{N\vec{a},g,r}$ from $\rho_{n,\ell}$. The advantage is ϵ and number of queries t.

Step I

Pick $v \in Z_N^*$ and compute $v^2 \bmod N$.

Step II (heart of the proof)

Using $\mathcal{M}$, construct a PPTM $\mathcal{D}$ s.t.:

$$\Pr\left[\mathcal{D}(v^2, r) = \mathcal{B}_r(u)\right] > \frac{1}{2} + \Omega\left(\frac{\epsilon}{t \cdot \ell}\right)$$

where $u \in Q_N$ such that $u^2 = v^2 \bmod N$.

Step III (Goldreich-Levin-Rackoff)

Using $\mathcal{D}$ as an oracle, reconstruct u, w.p. $\Omega\left(\frac{\epsilon^2}{t^2 \cdot \ell^2}\right)$.

Step IV (Rabin79)

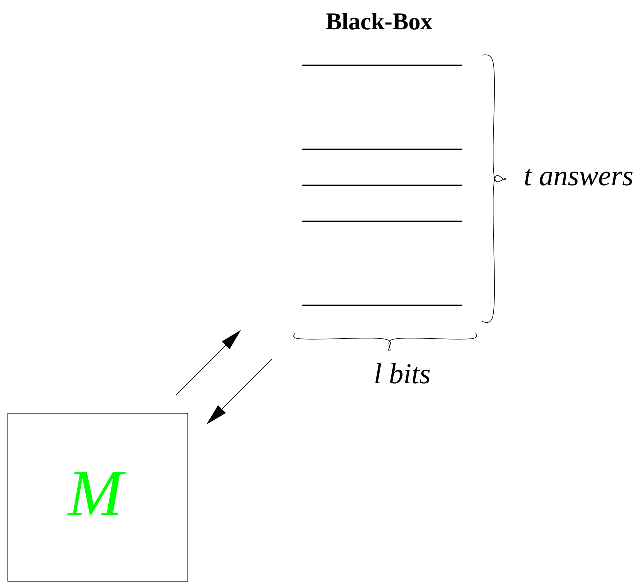
Since $u \in Q_N$, we have that:

$$\Pr\left[\gcd(u - v, N) \in \{P, Q\}\right] = \Pr\left[u \neq \pm v\right] = 1/2$$

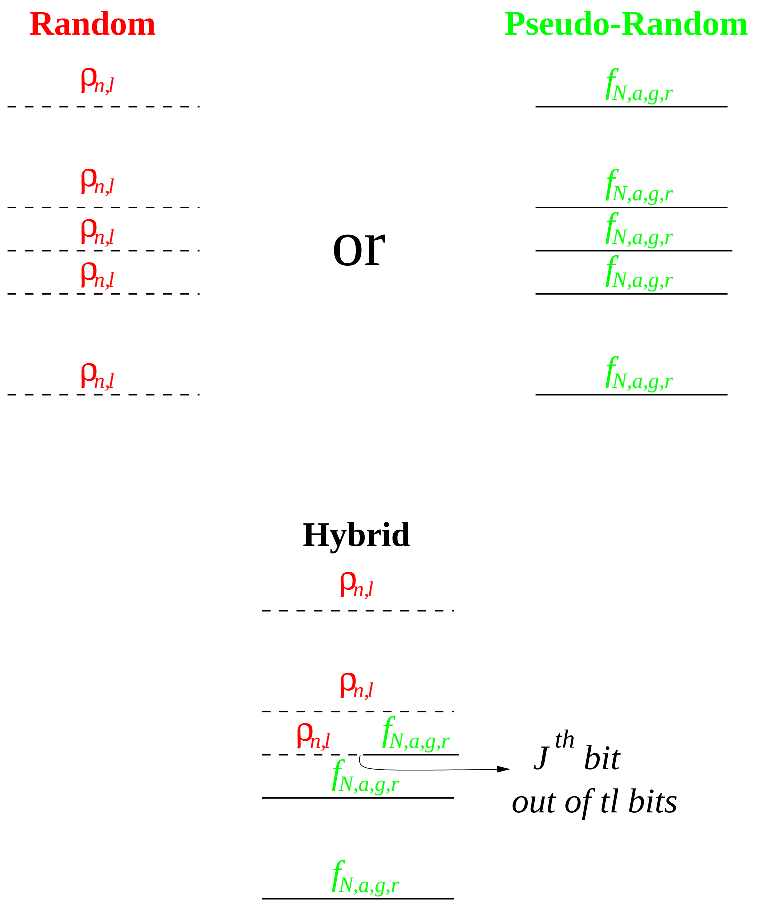
$\Rightarrow$ we can factor N with probability $\Omega\left(\frac{\epsilon^2}{t^2 \cdot \ell^2}\right)$.

Implementing Step II

Idea: $\mathcal{D}$ invokes $\mathcal{M}$ and simulates a black-box.



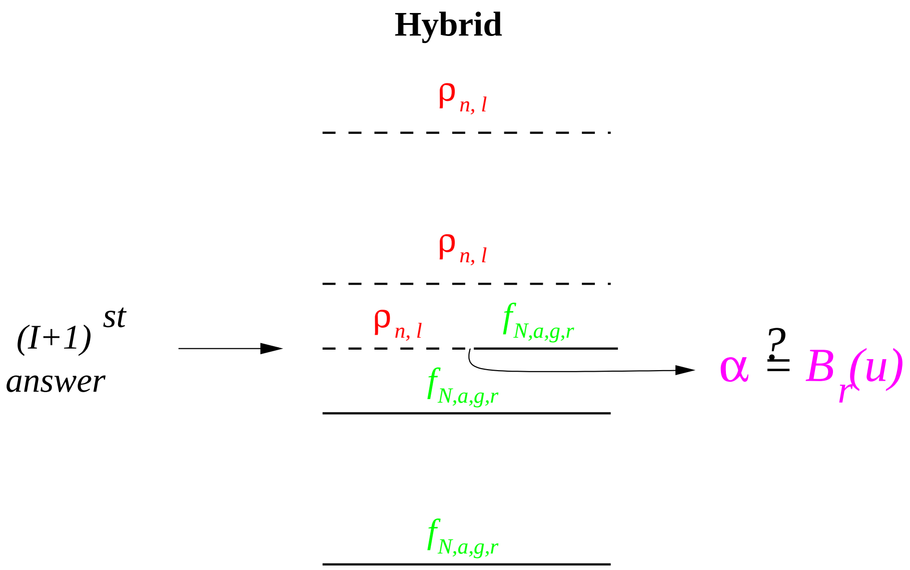
$\mathcal{D}$ simulates a hybrid black-box



Distinguishing $\mathcal{B}_r(u)$ from Random

$\mathcal{D}$ is given $\langle v^2, r, \alpha \rangle$.

Is α random?
Is α equal to $\mathcal{B}_r(u)$?



- If $\alpha = \mathcal{B}_r(u)$ then $\mathcal{M}$ "sees" J^{th} hybrid.
- If α is random then $\mathcal{M}$ "sees" $(J+1)^{st}$ hybrid.

$\Rightarrow$ $\mathcal{D}$ has advantage $\frac{\epsilon}{t \cdot \ell}$ in distinguishing $\mathcal{B}_r(u)$ from a random bit (since $\mathcal{M}$ has).

$\mathcal{D}$ simulates the hybrid black-box

$\mathcal{D}$ is given $\langle v^2, r, \alpha \rangle$.

1. $\mathcal{D}$ picks $J = I \cdot \ell + k$ at random.
   - I - number of random answers.
   - k - switching location in the $(I+1)^{st}$ answer.

2. $\mathcal{D}$ answers first I queries of $\mathcal{M}$ according to $\rho_{n,\ell}$.

Defining the key $\langle N, \vec{a}, g, r \rangle$:

3. Given $x = x_1 x_2 \ldots x_n$, the $(I+1)^{st}$ query of $\mathcal{M}$:

   - $g = v^{2^s}$ where $s = n\ell - k$.

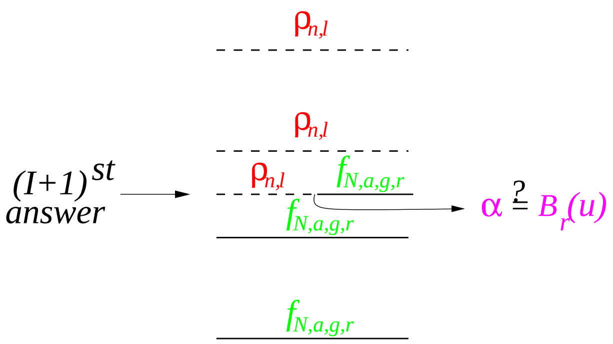
   - $\vec{a} = \left\langle \begin{matrix} a_{1,0}, & a_{2,0}, & & , a_{n,0} \\ a_{1,1}, & a_{2,1}, & \ldots & , a_{n,1} \end{matrix} \right\rangle$
     - $a_{i,x_i} = 2^{-\ell} \bmod ord_N(g)$.
     - $a_{i,\overline{x_i}} \in_R [N]$.

Claim: For every $i < s$, $g^{2^{-i}} = v^{2^{s-i}} \bmod N$

Simulation of $\mathcal{D}$ cont.

Corollary 2 Define $u = g^{2^{-s}}$, then $u^2 = v^2 \bmod N$.

Motivation:
$(k+1)^{st}$ bit in $(I+1)^{st}$ answer should be $\mathcal{B}_r(u)$.



4. $\mathcal{D}$ answers $(I+1)^{st}$ query with

$$b_0, b_1, \ldots, b_{k-1}, \alpha, \mathcal{B}_r(u^2), \ldots, \mathcal{B}_r(u^{2^{\ell-k-1}})$$

Note that $u^2 = v^2 \bmod N$
$\Rightarrow$ easy to compute $\mathcal{B}_r(u^2), \ldots, \mathcal{B}_r(u^{2^{\ell-k-1}})$.

5. $\mathcal{D}$ answers remaining queries according to $f_{N,\vec{a},g,r}$.

Answers should be consistent with $\langle N, \vec{a}, g, r\rangle$.

Correctness:

Recall that $\mathcal{D}$ answers $(I+1)^{st}$ query with

$$b_0, b_1, \ldots, b_{k-1}, \alpha, \mathcal{B}_r(u^2), \ldots, \mathcal{B}_r(u^{2^{\ell-k-1}})$$

According to the definition of $\langle N, \vec{a}, g, r \rangle$, $(k+1)^{st}$ bit in $(I+1)^{st}$ answer should be $\mathcal{B}_r(g^{2^k \prod_{i=1}^n a_{i,x_i}})$.

But:

$$g^{2^k \prod_{i=1}^n a_{i,x_i}} = g^{2^k \prod_{i=1}^n 2^{-\ell}} = g^{2^{k-\ell n}} = g^{2^{-s}} = u \bmod N$$

Therefore:

$(k+1)^{st}$ bit should be $\mathcal{B}_r(u)$, as required.

Answering remaining queries

If $y \neq x$ is a query, then:

$$
\begin{aligned}
& g^{\prod_{i=1}^{n} a_{i,y_i}} \\
= \; & g^{(\prod_{\{y_i = x_i\}} a_{i,y_i})(\prod_{\{y_i \neq x_i\}} a_{i,y_i})} \\
= \; & g^{(2^{-\ell j})(\prod_{\{y_i \neq x_i\}} a_{i,y_i})} \\
= \; & v^{(2^{s-\ell j})(\prod_{\{y_i \neq x_i\}} a_{i,y_i})} \bmod N
\end{aligned}
$$

j - number of locations for which $y_i = x_i$.

But:
Recall that $s = n\ell - k$.

$\Rightarrow \; s - \ell j = (n - j)\ell - k$ is at least 1 over Z.

Therefore:
$\mathcal{D}$ is able to compute $g^{\prod_{i=1}^{n} a_{i,y_i}}$ and consequently:

$$f_{N,\vec{a},g,r}(y) = G^{BBS}(g^{\prod_{i=1}^{n} a_{i,y_i}})$$

as required.

To randomize $\vec{a}$ set $a_{i,x_i} = 2^{-\ell} + \xi_i \bmod ord_N(g)$ (where $\xi_i \in_R [N]$).

Further Research

1. Recognize precise features of an "unpredictable" function and a pseudo-random generator which make our construction work.

2. Use same ideas for other "unpredictable" functions and pseudo-random generators.

3. Prove our construction using any hard-core predicate (or function) of squaring $\bmod N$.

4. Give alternative proof in which security does not decrease proportionally to the number of queries made by $\mathcal{M}$ (i.e. t).

5. DDH-based functions have zero-knowledge, oblivious function evaluation and sharing protocols.

What Are Good Assumptions?

Assumption form: no adverserial machine running in time T can succeed in breaking the assumption with probability greater than ϵ.

Good assumptions:

- Criterion: assumption should be refutable. I.e. there must be a way to demonstrate that it is false, *if* this is the case.
- Should take into account the complexity of refutation, especially verification.

Best way to refute:

- The assumption's author publishes a set of challenges
- The refuter sends the solution
- Alarm should sound.

Definition of the challenge process should be part of the design of the cryptosystem!

Efficient Refutation

Efficiently Refutable: the size of the challenge and the time for verification depends only polylog. on T and on $1/\epsilon$.

If the assumption is false, then there is a machine M that solves the challenge in time $(poly)(T, \epsilon)$ whp $(> \delta)$.

Verifier publishes a challenge and does not need to simulate adversary.

Comment: (run time $< T$) is not verified.

Efficient Refutation - Examples

A function f is one-way if for random x it is hard to find $x' \in f^{-1}(x)$.

Assumption: function f cannot be inverted in time T with probability of success better than ϵ.

One wayness is efficiently refutable for

- If f is a permutation - $\surd$
- General f - ?

Related to amplification!

Factoring is hard - efficiently refutable

- Blum integers are not sparse
- The challenge - a range, *result of hashing*.
  Any Blum integer factored in the range - a refutation!