

Efficient Pseudorandom Generators Based on the DDH Assumption

Andrey Sidorenko

(Joint work with Reza Rezaeian Farashahi and Berry Schoenmakers)

TU Eindhoven

Outline

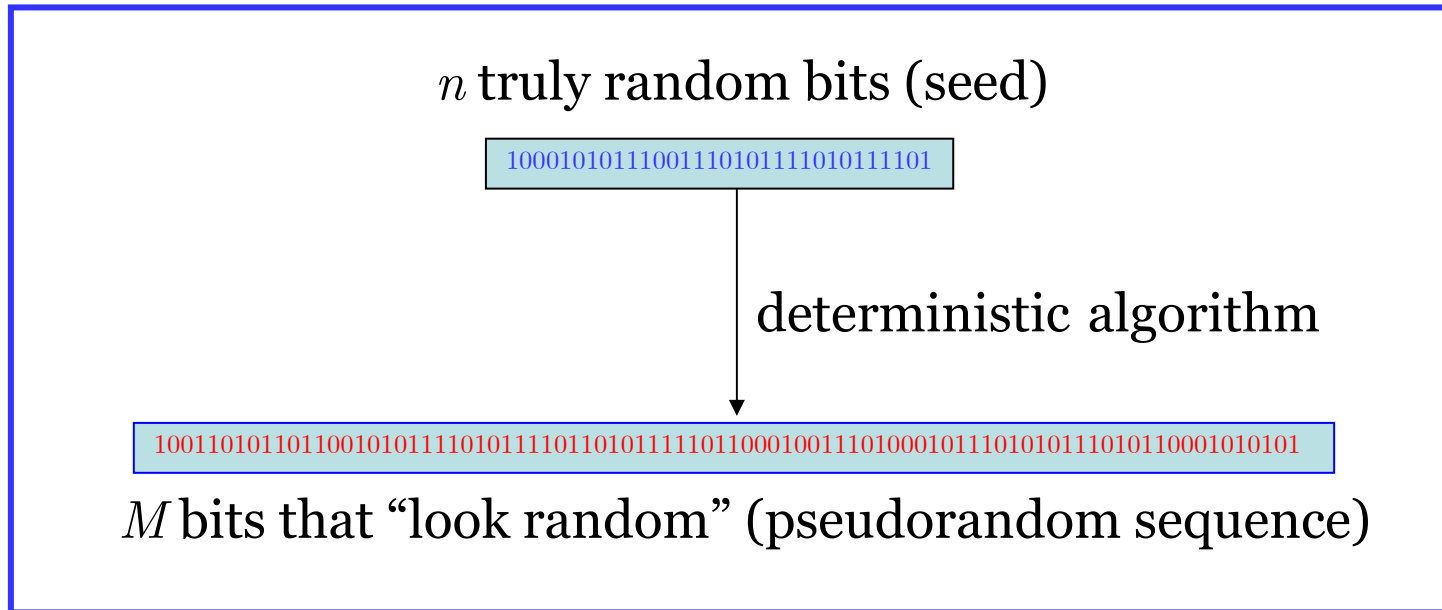
- Introduction
 - provably secure pseudorandom generators (PRG)
 - Blum-Micali PRG
 - based on the DL problem
 - security reduction is not tight
 - decisional Diffie-Hellman (DDH) problem
 - **concrete** vs. asymptotic security
- New PRG based on the DDH problem
 - **almost tight reduction**
 - specific instances
 - group of quadratic residues modulo safe prime
 - arbitrary subgroup of Z_p^*

Application of randomness

- Noncryptographic purposes
 - Monte Carlo Simulations (collecting statistics)
 - Randomized algorithms (e.g. QuickSort)
- Cryptographic schemes
 - Probabilistic signing/encryption algorithms
 - Protocols, using
 - Nonces
 - Session keys
 - Stream Ciphers
 - Lotteries and electronic gambling
- Stronger constraints for cryptographic use!!
 - *Security \approx indistinguishability*



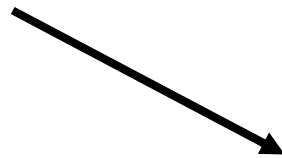
Pseudorandom Generator



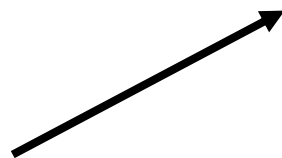
- Preferably $M \gg n$ and fast PRG
- Seed value is usually derived from variances in timing of low-level devices
 - keyboard, mouse, disc drive, process ID

Security of PRGs

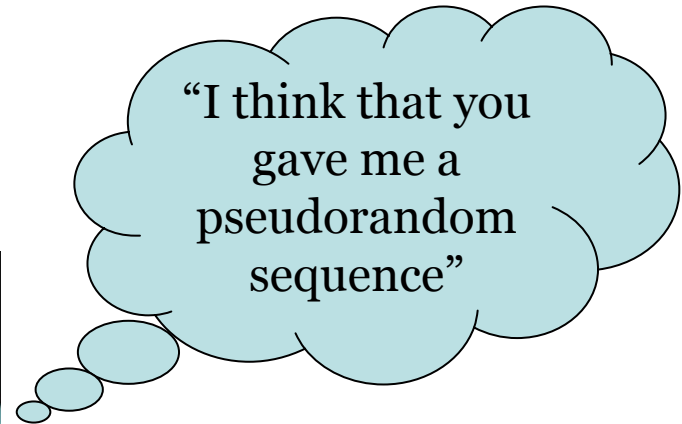
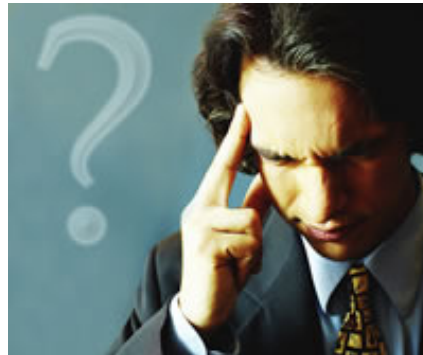
Pseudorandom sequence
chosen at random



OR



Distinguisher:
running time T



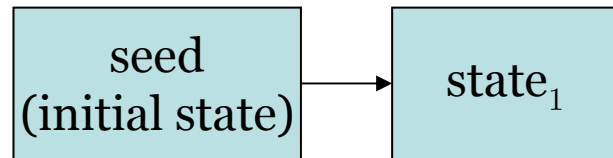
Truly random sequence of
the same length

If probability of successful
guess $< \frac{1}{2}(1 + \varepsilon)$ the PRG is
called (T, ε) -secure

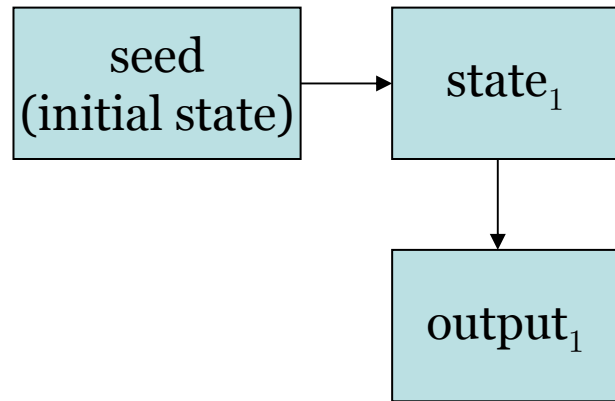
“Typical PRG”

seed
(initial state)

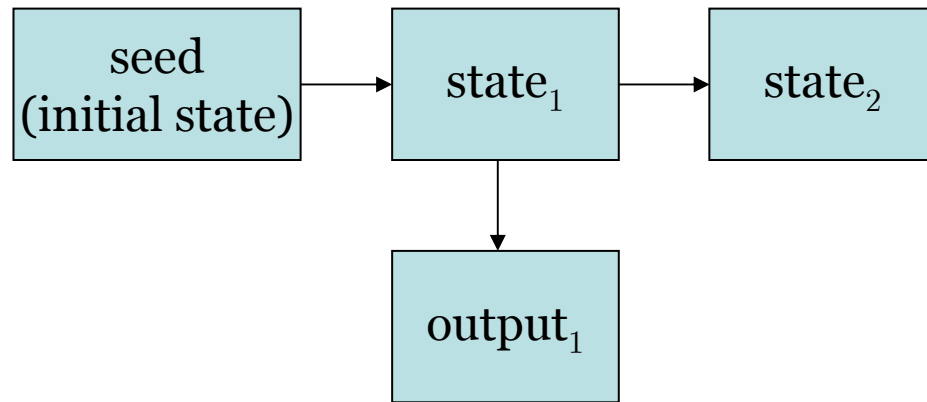
“Typical PRG”



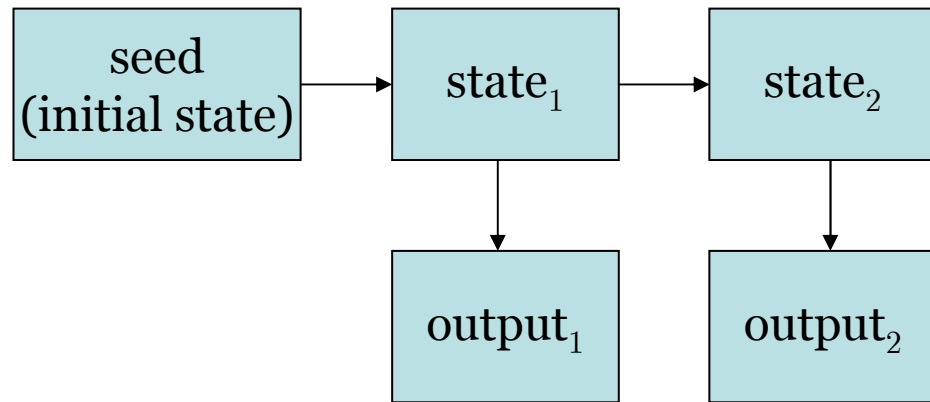
“Typical PRG”



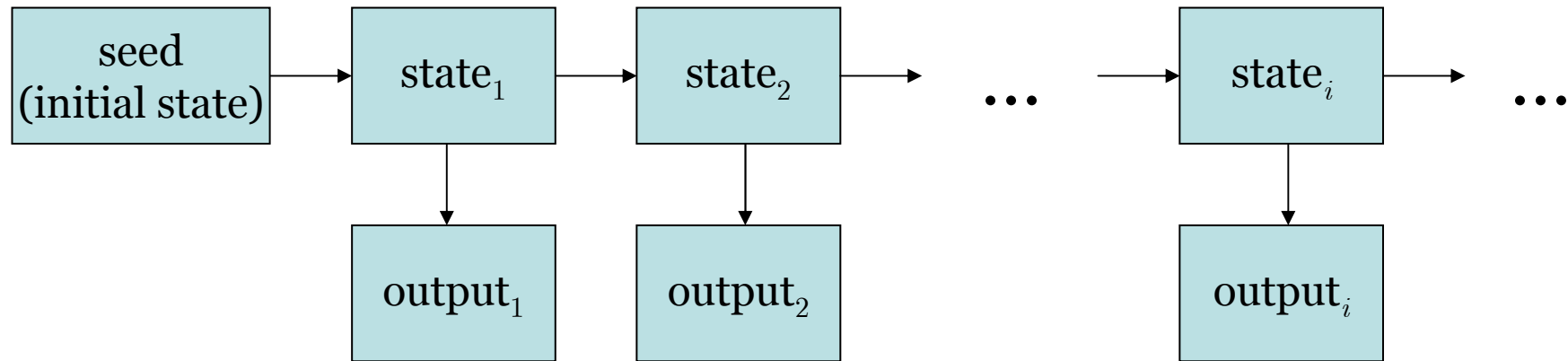
“Typical PRG”



“Typical PRG”



“Typical PRG”



Some PRGs used in practice

- function rand()
 - included into the ANSI C Standard
 - uses a 32-bit seed
 - based on a linear feedback shift register
 - non-cryptographic
- class SHA1PRNG
 - included into an open source Java library
 - uses 160-bit seed
 - based on SHA-1 hash function
 - seems to be secure

Some PRGs used in practice

- function rand()
 - included into the ANSI C Standard
 - uses a 32-bit seed
 - based on a linear feedback shift register
 - non-cryptographic
- class SHA1PRNG
 - included into an open source Java library
 - uses 160-bit seed
 - based on SHA-1 hash function
 - seems to be secure
 - SHA-1 is not collision-resistant [Wang et al. 2005]
 - does it affect security of the generator?

Some PRGs used in practice

- function rand()
 - included into the ANSI C Standard
 - uses a 32-bit seed
 - based on a linear feedback shift register
 - **non-cryptographic**
- class SHA1PRNG
 - included into an open source Java library
 - uses 160-bit seed
 - based on SHA-1 hash function
 - seems to be secure
 - **SHA-1 is not collision-resistant [Wang et al. 2005]**
 - **does it affect security of the generator?**
- ***What about provably secure PRGs?***
 - a PRG is called provably secure if “breaking” the PRG is as hard as solving a difficult (unsolvable?) problem

Provably secure PRGs

(T, ε) -distinguisher for a
PRG: $\{0, 1\}^n \rightarrow \{0, 1\}^M$



(T', ε') -solver for a hard
problem with security
parameter n
(e.g., DL problem in n -bit
finite field)

- If $T'/\varepsilon' \approx T/\varepsilon$, the reduction is called **tight**
- If $T'/\varepsilon' \gg T/\varepsilon$, the reduction is called **not tight**

Provably secure PRGs

(T, ε) -distinguisher for a
PRG: $\{0, 1\}^n \rightarrow \{0, 1\}^M$



(T', ε') -solver for a hard
problem with security
parameter n
(e.g., DL problem in n -bit
finite field)

- If $T'/\varepsilon' \approx T/\varepsilon$, the reduction is called **tight**
- If $T'/\varepsilon' \gg T/\varepsilon$, the reduction is called **not tight**
- Tighter reduction► desired security level
for smaller value of n

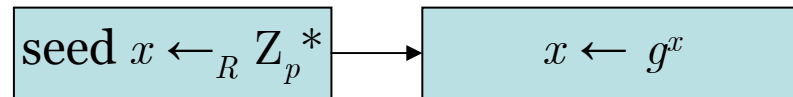
Blum-Micali PRG [1982]

Let g be a generator of Z_p^*

seed $x \leftarrow_R Z_p^*$

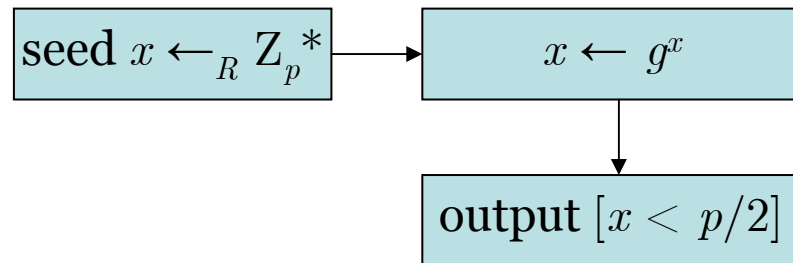
Blum-Micali PRG [1982]

Let g be a generator of Z_p^*



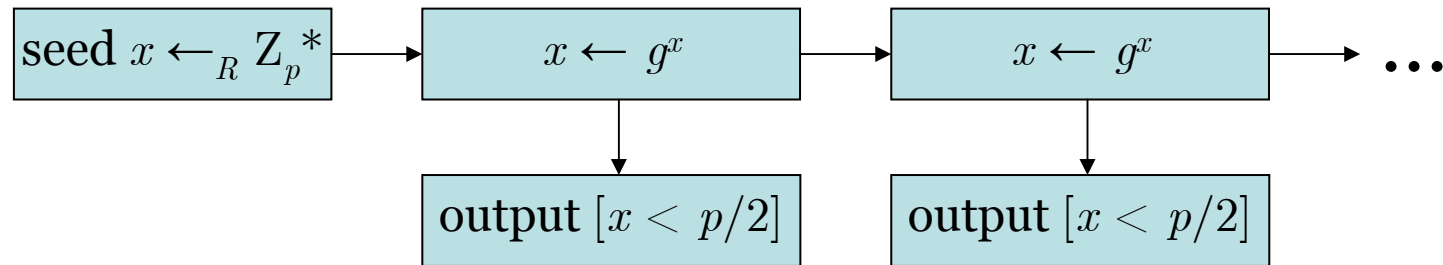
Blum-Micali PRG [1982]

Let g be a generator of Z_p^*



Blum-Micali PRG [1982]

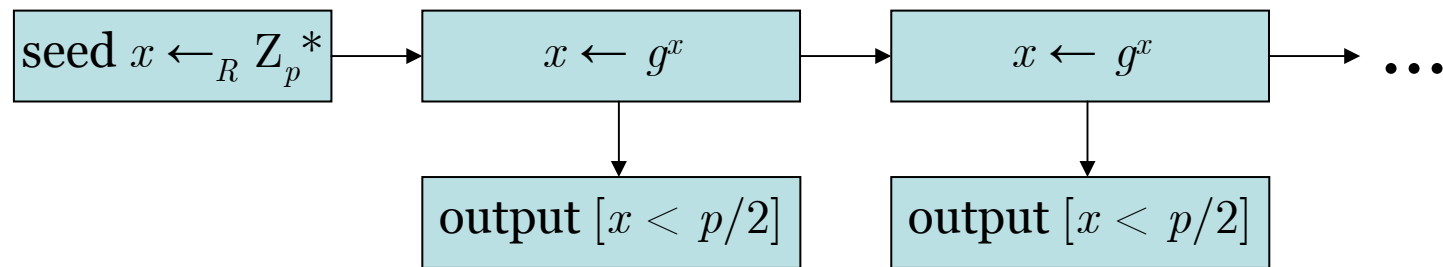
Let g be a generator of Z_p^*



- Outputs only 1 bit per modular exponentiation

Blum-Micali PRG [1982]

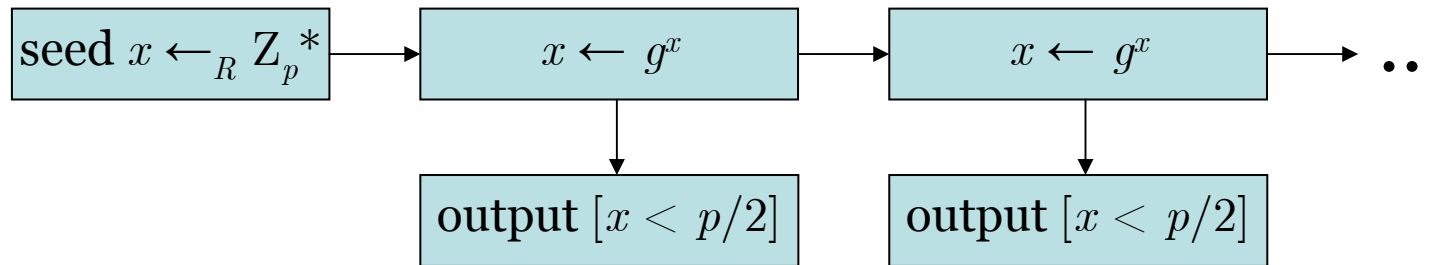
Let g be a generator of Z_p^*



- Outputs only 1 bit per modular exponentiation
- **Security proof:** (T, ε) -distinguisher implies a solver for DL problem with expected time $T' = 128n^3(M/\varepsilon)^4T$
 - $n = \log_2 p$
 - M is total number of output bits

Blum-Micali PRG [1982]

Let g be a generator of Z_p^*



- Outputs only 1 bit per modular exponentiation
- **Security proof:** (T, ε) -distinguisher implies a solver for DL problem with expected time $T' = 128n^3(M/\varepsilon)^4T$
 - $n = \log_2 p$
 - M is total number of output bits
- $128n^3(M/\varepsilon)^4$ is a large factor, so the reduction is **not tight**

Blum-Micali PRG (continued)

- Thus, the Blum-Micali generator is (T, ε) -secure if

$$128 n^3 (M/\varepsilon)^4 T < T_{\text{DL}}(\mathbb{Z}_p^*),$$

where $T_{\text{DL}}(\mathbb{Z}_p^*)$ is estimated time to solve the DL problem in \mathbb{Z}_p^*

Blum-Micali PRG (continued)

- Thus, the Blum-Micali generator is (T, ε) -secure if

$$128 n^3 (M/\varepsilon)^4 T < T_{\text{DL}}(\mathbb{Z}_p^*),$$

where $T_{\text{DL}}(\mathbb{Z}_p^*)$ is estimated time to solve the DL problem in \mathbb{Z}_p^*

- For $M = 2^{20}$, $T/\varepsilon = 2^{80}$, BM PRG is (T, ε) -secure if $n > 61000$
- High seed length implies poor efficiency
 - the reason is that **the reduction is not tight**

Blum-Micali PRG (continued)

- Thus, the Blum-Micali generator is (T, ε) -secure if

$$128 n^3 (M/\varepsilon)^4 T < T_{\text{DL}}(\mathbb{Z}_p^*),$$

where $T_{\text{DL}}(\mathbb{Z}_p^*)$ is estimated time to solve the DL problem in \mathbb{Z}_p^*

- For $M = 2^{20}$, $T/\varepsilon = 2^{80}$, BM PRG is (T, ε) -secure if $n > 61000$
- High seed length implies poor efficiency
 - the reason is that **the reduction is not tight**
- We propose a PRG with much tighter security reduction
 - based on the DDH assumption (stronger than DL assumption)
 - output n bits per iteration

Decisional Diffie-Hellman (DDH) problem

G multiplicative group of prime order q
– g generator of G

- Algorithm A solves the DDH problem in G with advantage ε if and only if for a random triple (a, b, r)

$$| \Pr(A(g, g^a, g^b, g^{ab}) = 1) - \Pr(A(g, g^a, g^b, g^r) = 1) | \geq \varepsilon$$

- We consider groups in which DDH is assumed to be as hard as DL

DDH generator (intuition)

G multiplicative group of prime order q
– g generator of G

- $x, y \leftarrow_R G$
- Let **Double** $_{x,y}(s) = (x^s, y^s)$ [Naor, Reingold 1997]
 - for unknown $s \leftarrow_R \mathbb{Z}_q$ the output is pseudorandom under the DDH assumption in G
 - “doubles” the input
- Is **Double** a pseudorandom generator?

DDH generator (intuition)

G multiplicative group of prime order q
– g generator of G

- $x, y \leftarrow_R G$
- Let **Double** $_{x,y}(s) = (x^s, y^s)$ [Naor, Reingold 1997]
 - for unknown $s \leftarrow_R \mathbb{Z}_q$ the output is pseudorandom under the DDH assumption in G
 - “doubles” the input
- Is **Double** a pseudorandom generator?
 - **No!** It produces pseudorandom group elements rather than pseudorandom bits
 - Converting group elements into bits is a non-trivial problem
 - **Double cannot be iterated** to produce as much randomness as required by the application

DDH generator (construction)

- **enum** is a bijective function that “enumerates” the elements of the group

$$\text{enum: } G \times Z_l \rightarrow Z_q \times Z_l$$

DDH generator (construction)

- **enum** is a bijective function that “enumerates” the elements of the group

$$\text{enum: } G \times Z_l \rightarrow Z_q \times Z_l$$

- Public parameters $x, y \leftarrow_R G$

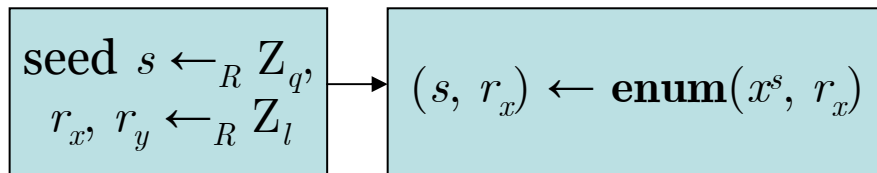
seed $s \leftarrow_R Z_q,$ $r_x, r_y \leftarrow_R Z_l$

DDH generator (construction)

- **enum** is a bijective function that “enumerates” the elements of the group

$$\text{enum}: G \times Z_l \rightarrow Z_q \times Z_l$$

- Public parameters $x, y \leftarrow_R G$

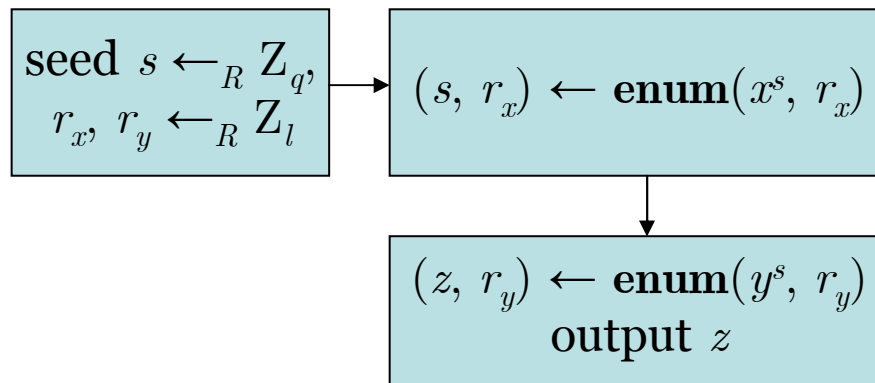


DDH generator (construction)

- **enum** is a bijective function that “enumerates” the elements of the group

$$\text{enum}: G \times Z_l \rightarrow Z_q \times Z_l$$

- Public parameters $x, y \leftarrow_R G$

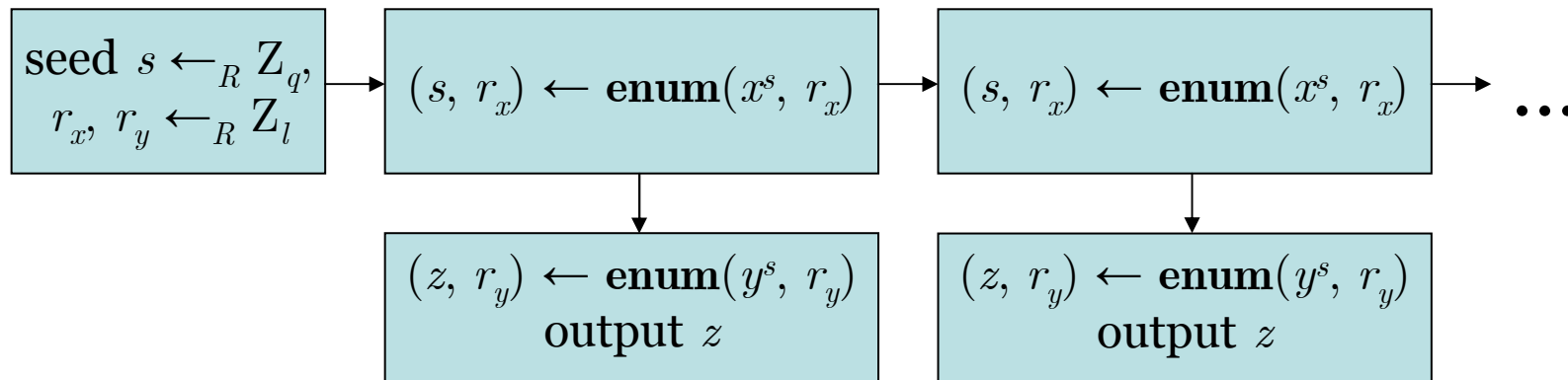


DDH generator (construction)

- `enum` is a bijective function that “enumerates” the elements of the group

$$\text{enum}: G \times Z_l \rightarrow Z_q \times Z_l$$

- Public parameters $x, y \leftarrow_R G$



Security of the DDH generator

- DDH generator produces pseudorandom integers from Z_q
 - if $q \approx 2^n$, it produces pseudorandom bits
 - for an arbitrary q , convert random numbers into random bits (easy problem)

Security of the DDH generator

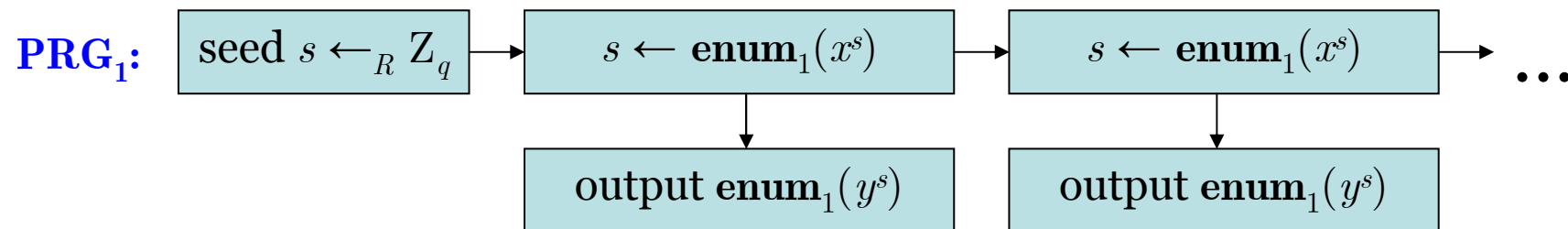
- DDH generator produces pseudorandom integers from Z_q
 - if $q \approx 2^n$, it produces pseudorandom bits
 - for an arbitrary q , convert random numbers into random bits (easy problem)
- **Security proof:** reduction of (T, ε) -distinguisher to a solver for DDH problem in G with time T with advantage ε/k
 - k is the number of output integers
 - proof is based on the hybrid argument
 - reduction is almost tight

First instance

- p is a safe prime, that is, $p = 2q + 1$, q is prime
- $G = \text{QR}_p$, $|G| = q$
- Bijection from G to Z_q (see, e.g., Cramer-Shoup 2003; Chevassut et al. 2005):

$$\mathbf{enum}_1(x) = \begin{cases} x, & \text{if } 1 \leq x \leq q; \\ p - x, & \text{if } q + 2 \leq x < p; \\ 0, & \text{otherwise.} \end{cases}$$

- Public parameters $x, y \leftarrow_R G$



- Extracts $\log_2 p$ bits per 2 modular exponentiation

First instance – performance

- Let $M = 2^{20}$, $T/\varepsilon = 2^{80}$
 - Corresponds to solving DL problem in 1200-bit fields
- What seed length n guarantees security?

First instance – performance

- Let $M = 2^{20}$, $T/\varepsilon = 2^{80}$
 - Corresponds to solving DL problem in 1200-bit fields
- What seed length n guarantees security?
- Recall that for Blum-Micali PRG $n > 61000$
 - Huge prime p , more than 50 times 1200

First instance – performance

- Let $M = 2^{20}$, $T/\varepsilon = 2^{80}$
 - Corresponds to solving DL problem in 1200-bit fields
- What seed length n guarantees security?
- Recall that for Blum-Micali PRG $n > 61000$
 - Huge prime p , more than 50 times 1200
- **PRG₁** is (T, ε) -secure if... $n > 1600$
 - Only 30% larger than standard 1200 bit modulus

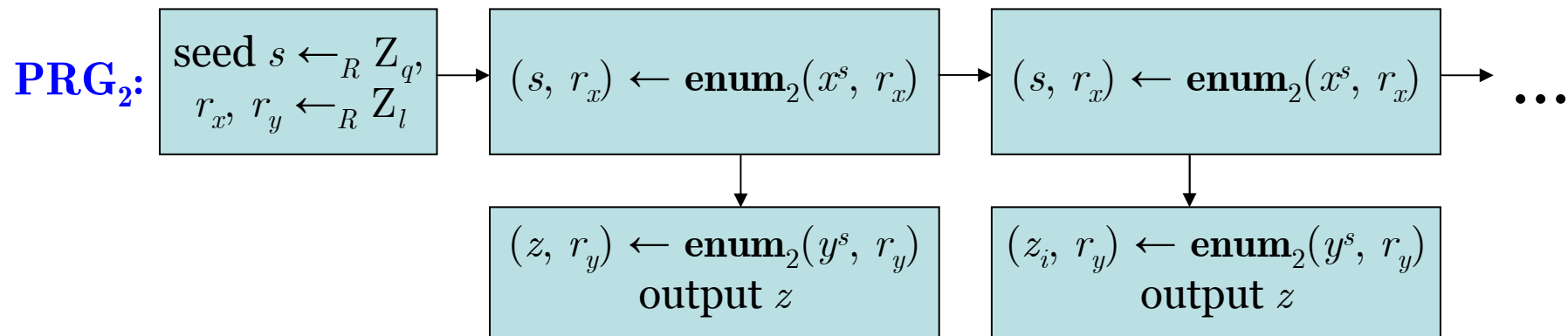
First instance – performance

- Let $M = 2^{20}$, $T/\varepsilon = 2^{80}$
 - Corresponds to solving DL problem in 1200-bit fields
- What seed length n guarantees security?
- Recall that for Blum-Micali PRG $n > 61000$
 - Huge prime p , more than 50 times 1200
- **PRG₁** is (T, ε) -secure if... $n > 1600$
 - Only 30% larger than standard 1200 bit modulus
- The secure seed length is short because the reduction is (almost) tight
 - **PRG₁** is much more efficient than Blum-Micali PRG
 - **PRG₁** is based on a stronger assumption (the DDH assumption)
 - Limitation: works only for specific subgroup of Z_p^*

Second instance

- p is a prime
- G is an arbitrary prime order subgroup of Z_p^* , $|G| = q$, $(p - 1) = ql$
- $t \leftarrow Z_p^*$ is an element of order l , that is, $t^l = 1$
- Bijection from $G \times Z_l$ to $Z_q \times Z_l$:

$$\text{enum}_2(x, r) = (x t^r \bmod q, x t^r \text{ div } q)$$
- Public parameters $x, y \leftarrow_R G$

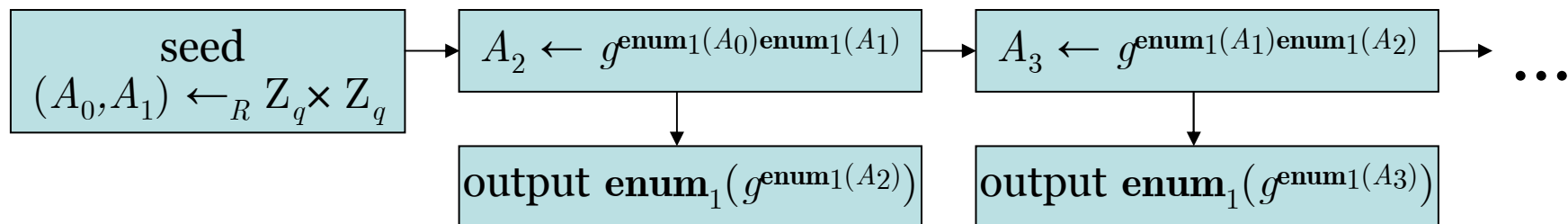


Related work

- Independently, Jiang [ACISP 2006] proposed another PRG, which is also provably secure under the DDH assumption
- g is a generator of QR_p , where $p = 2q + 1$ is a safe prime, $\#\text{QR}_p = q$
- As before,

$$\mathbf{enum}_1(x) = \begin{cases} x, & \text{if } 1 \leq x \leq q; \\ p - x, & \text{if } q + 2 \leq x < p; \\ 0, & \text{otherwise.} \end{cases}$$

Jiang's generator:



Jiang's generator versus ours

- The speed of the two generators is the same
- Our generator can be used with *any* prime order group such that the elements of the group can be “efficiently enumerated”
 - Jiang's generator is designed for a *specific* subgroup of Z_p^* for *safe primes* p
- The seed of our generator is *two times shorter* than the seed of Jiang's generator
 - For most of the applications, the length of the seed is a critical issue

Conclusion

- A new pseudorandom generator based on the decisional Diffie-Hellman assumption is proposed
- Two specific instances of the new PRG are presented
 - subgroup of quadratic residues modulo safe prime
 - arbitrary subgroup of Z_p^*
- Open problem: design a pseudorandom generator based on the DDH problem in an ordinary elliptic curve