Easing Coppersmith Methods using Analytic Combinatorics: Applications to Public-Key Cryptography with Weak Pseudorandomness

Fabrice Benhamouda, Céline Chevalier, Adrian Thillard, and Damien Vergnaud

École normale supérieure, CNRS, INRIA, PSL, Université Panthéon-Assas, ANSSI, Paris, France

PKC 2016, Taipei, Taiwan

Coppersmith Methods

Quick History

- Introduced by Coppersmith in 1996 to find:
  - small roots of univariate modular polynomials [Cop96b];
  - small roots of bivariate polynomials [Cop96a];

Coppersmith Methods

Quick History

- Introduced by Coppersmith in 1996 to find:
  - small roots of univariate modular polynomials [Cop96b];
    - e.g., decrypt RSA with known plaintext MSB β:

$$(2^k \cdot \beta + x)^e \bmod N = c \quad \text{with } |x| \leq 2^k$$

  - small roots of bivariate polynomials [Cop96a];

Coppersmith Methods

Quick History

- Introduced by Coppersmith in 1996 to find:
  - small roots of univariate modular polynomials [Cop96b];
    - e.g., decrypt RSA with known plaintext MSB β:

$$(2^k \cdot \beta + x)^e \bmod N = c \quad \text{with } |x| \leq 2^k$$

    - extension of small plaintext: $x^e \bmod N = c$;
  - small roots of bivariate polynomials [Cop96a];

Coppersmith Methods

Quick History

- Introduced by Coppersmith in 1996 to find:
  - small roots of univariate modular polynomials [Cop96b];
    - e.g., decrypt RSA with known plaintext MSB β:

$$(2^k \cdot \beta + x)^e \bmod N = c \quad \text{with } |x| \leq 2^k$$

    - extension of small plaintext: $x^e \bmod N = c$;
  - small roots of bivariate polynomials [Cop96a];
    - e.g., factorizing with known primes MSB:

$$(2^k \cdot \alpha + x) \cdot (2^k \cdot \beta + y) = N \quad \text{with } |x|, |y| \leq 2^k$$

Coppersmith Methods

Quick History

- Introduced by Coppersmith in 1996 to find:
  - small roots of univariate modular polynomials [Cop96b];
    - e.g., decrypt RSA with known plaintext MSB β:

$$(2^k \cdot \beta + x)^e \bmod N = c \quad \text{with } |x| \leq 2^k$$

    - extension of small plaintext: $x^e \bmod N = c$;
  - small roots of bivariate polynomials [Cop96a];
    - e.g., factorizing with known primes MSB:

$$(2^k \cdot \alpha + x) \cdot (2^k \cdot \beta + y) = N \quad \text{with } |x|, |y| \leq 2^k$$

- Further extensions:
  - more variables [HG97, BM05, JM06];
  - multiple polynomials and moduli [MR08, MR09, Rit10].

Coppersmith Methods

Quick History

- Introduced by Coppersmith in 1996 to find:
  - small roots of univariate modular polynomials [Cop96b];
    - e.g., decrypt RSA with known plaintext MSB β:

$$(2^k \cdot \beta + x)^e \bmod N = c \quad \text{with } |x| \leq 2^k$$

    - extension of small plaintext: $x^e \bmod N = c$;
  - small roots of bivariate polynomials [Cop96a];
    - e.g., factorizing with known primes MSB:

$$(2^k \cdot \alpha + x) \cdot (2^k \cdot \beta + y) = N \quad \text{with } |x|, |y| \leq 2^k$$

- Further extensions:
  - more variables [HG97, BM05, JM06];
  - multiple polynomials and moduli [MR08, MR09][Rit10].

Coppersmith Methods

Goal

Solve:

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0 \mod N_1 \\ \vdots \\ f_s(x_1, \ldots, x_n) = 0 \mod N_s \end{cases}$$

with

$$|x_1| \leq X_1 \qquad \ldots \qquad |x_n| \leq X_n$$

Coppersmith Methods

Goal

Solve:

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0 \mod N_1 \\ \vdots \\ f_s(x_1, \ldots, x_n) = 0 \mod N_s \end{cases}$$

with

$$|x_1| \leq X_1 \qquad \ldots \qquad |x_n| \leq X_n$$

Question: which bounds $X_1, \ldots, X_n$ work?

Coppersmith Methods

Overview

1. Construction of polynomials $\tilde{f}_{i,j}$ such that:

$$\tilde{f}_{i,j}(x_1, \ldots, x_n) = 0 \mod N_i^{k_{i,j}}$$

for any original solution $(x_1, \ldots, x_n)$.

Coppersmith Methods

Overview

1. Construction of polynomials $\tilde{f}_{i,j}$ such that:

$$\tilde{f}_{i,j}(x_1, \ldots, x_n) = 0 \mod N_i^{k_{i,j}}$$

for any original solution $(x_1, \ldots, x_n)$.

2. Use LLL to find an integer system:

$$\begin{cases} g_1(x_1, \ldots, x_n) = 0 \\ \vdots \\ g_n(x_1, \ldots, x_n) = 0 \end{cases}$$

such that:

1. any original solution is satisfied;
2. it has only a finite number of solutions.

Coppersmith Methods

Overview

1. Construction of polynomials $\tilde{f}_{i,j}$ such that:

$$\tilde{f}_{i,j}(x_1, \ldots, x_n) = 0 \mod N_i^{k_{i,j}}$$

for any original solution $(x_1, \ldots, x_n)$.

2. Use LLL to find an integer system:

$$\begin{cases} g_1(x_1, \ldots, x_n) = 0 \\ \vdots \\ g_n(x_1, \ldots, x_n) = 0 \end{cases}$$

such that:
   1. any original solution is satisfied;
   2. it has only a finite number of solutions.

3. Solve the system (using Groebner basis).

Coppersmith Methods

Condition and Combinatorics

- Success condition = combinatorial condition on
  - the number of polynomials $\tilde{f}_{i,j}$
  - the number of monomials in $\tilde{f}_{i,j}$
  - the moduli $N_i^{k_i,j}$
  - the bounds X_i

Coppersmith Methods
Condition and Combinatorics

- Success condition = combinatorial condition on
  - the number of polynomials $\tilde{f}_{i,j}$
  - the number of monomials in $\tilde{f}_{i,j}$
  - the moduli $N_i^{k_i,j}$
  - the bounds X_i

- Complexity: idem

Coppersmith Methods
Condition and Combinatorics

- Success condition = combinatorial condition on
  - the number of polynomials $\tilde{f}_{i,j}$
  - the number of monomials in $\tilde{f}_{i,j}$
  - the moduli $N_i^{k_i,j}$
  - the bounds X_i

- Complexity: idem

Difficult to compute when s and n non-constant

Coppersmith Methods
Condition and Combinatorics

- Success condition = combinatorial condition on
  - the number of polynomials $\tilde{f}_{i,j}$
  - the number of monomials in $\tilde{f}_{i,j}$
  - the moduli $N_i^{k_i,j}$
  - the bounds X_i

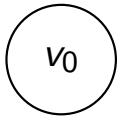
- Complexity: idem

Difficult to compute when s and n non-constant

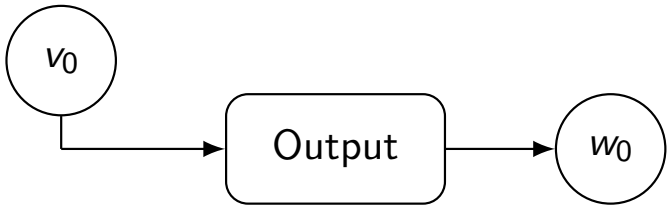
Our solution

Use analytic combinatorics!

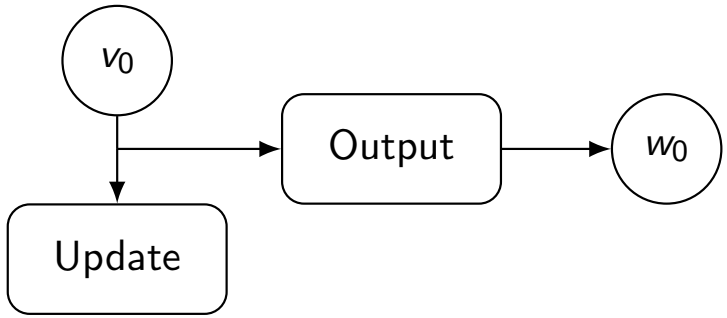
Pseudorandom Generator (PRG)



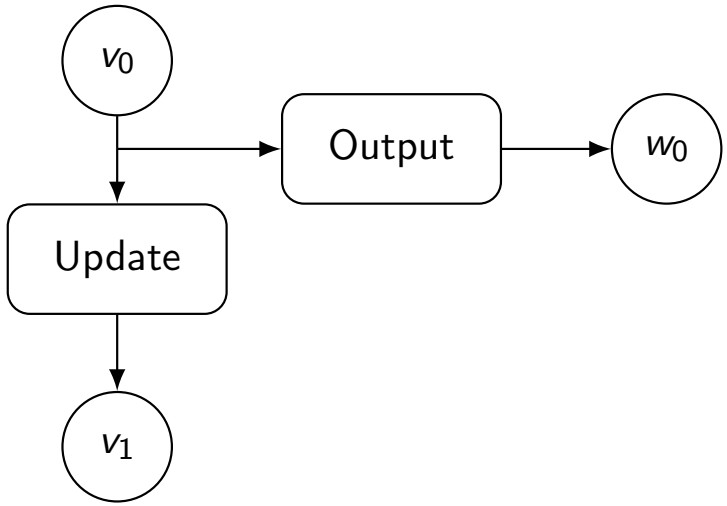
Pseudorandom Generator (PRG)



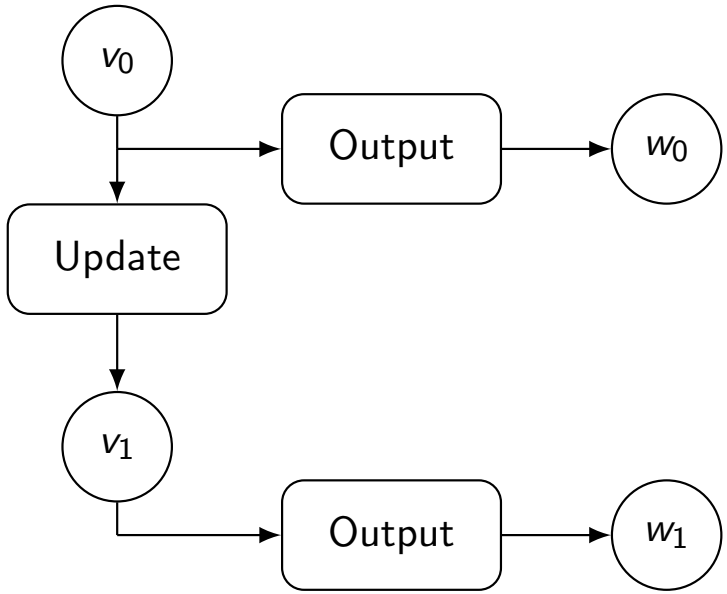
Pseudorandom Generator (PRG)



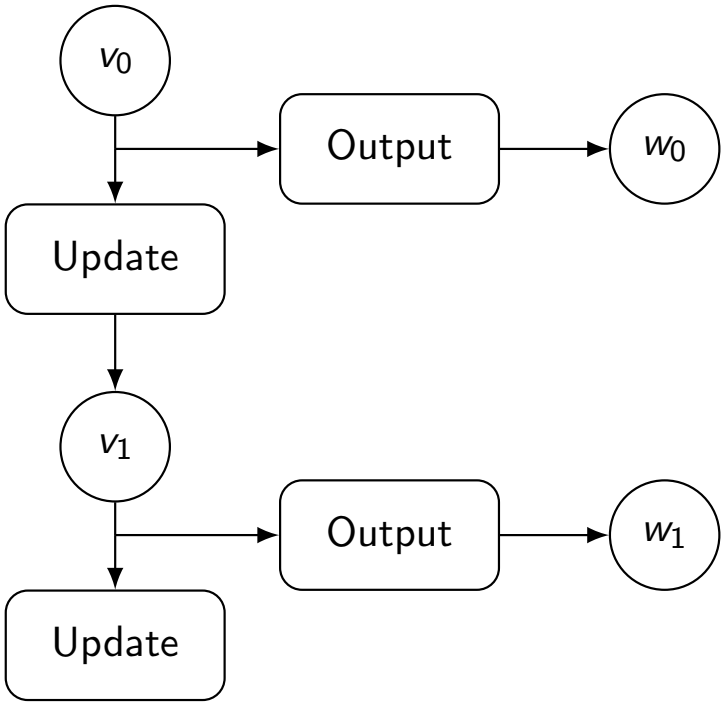
Pseudorandom Generator (PRG)



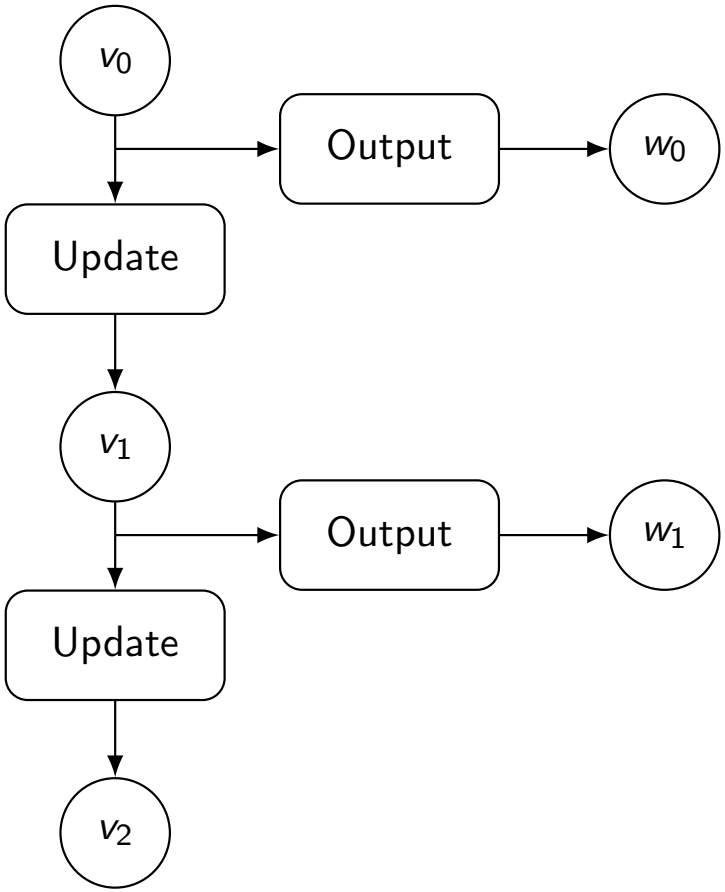
Pseudorandom Generator (PRG)



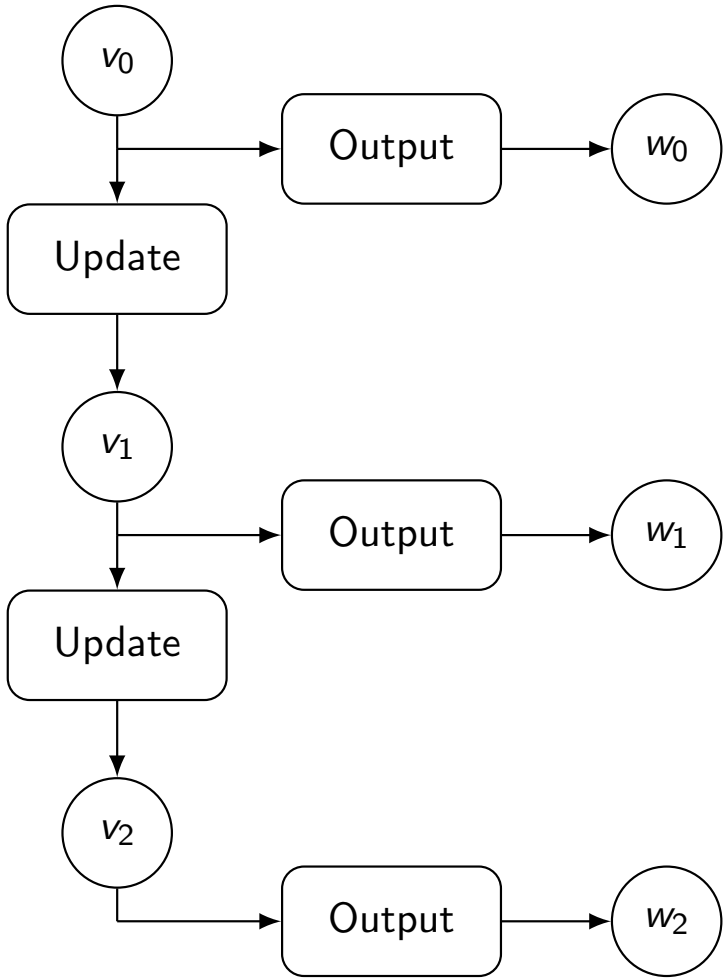
Pseudorandom Generator (PRG)



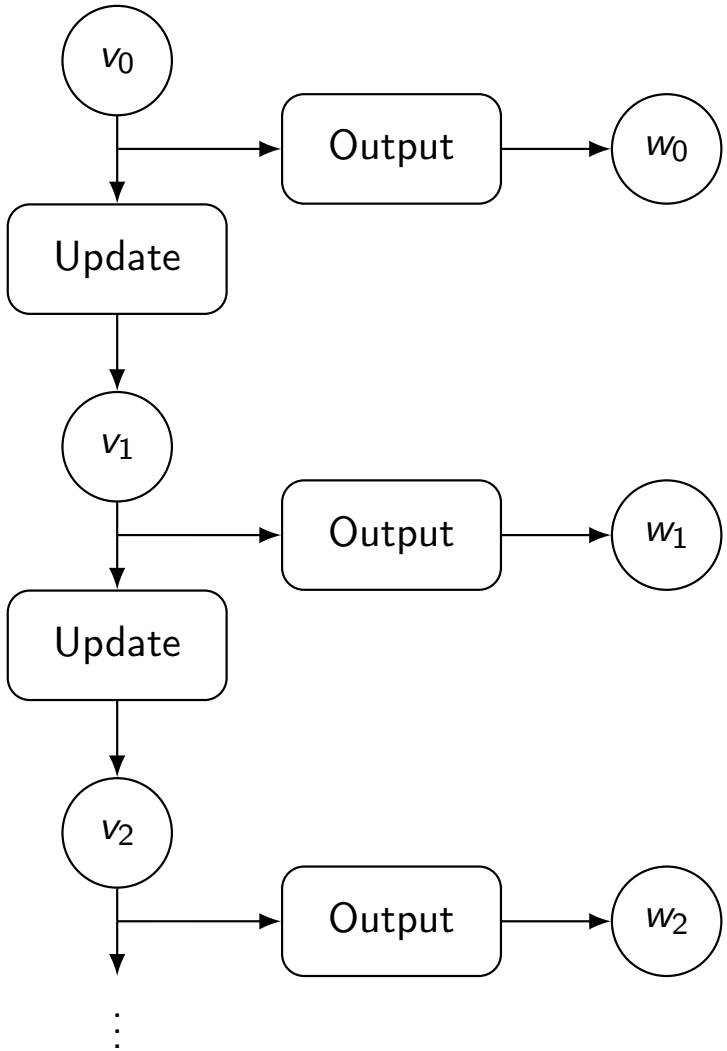
Pseudorandom Generator (PRG)



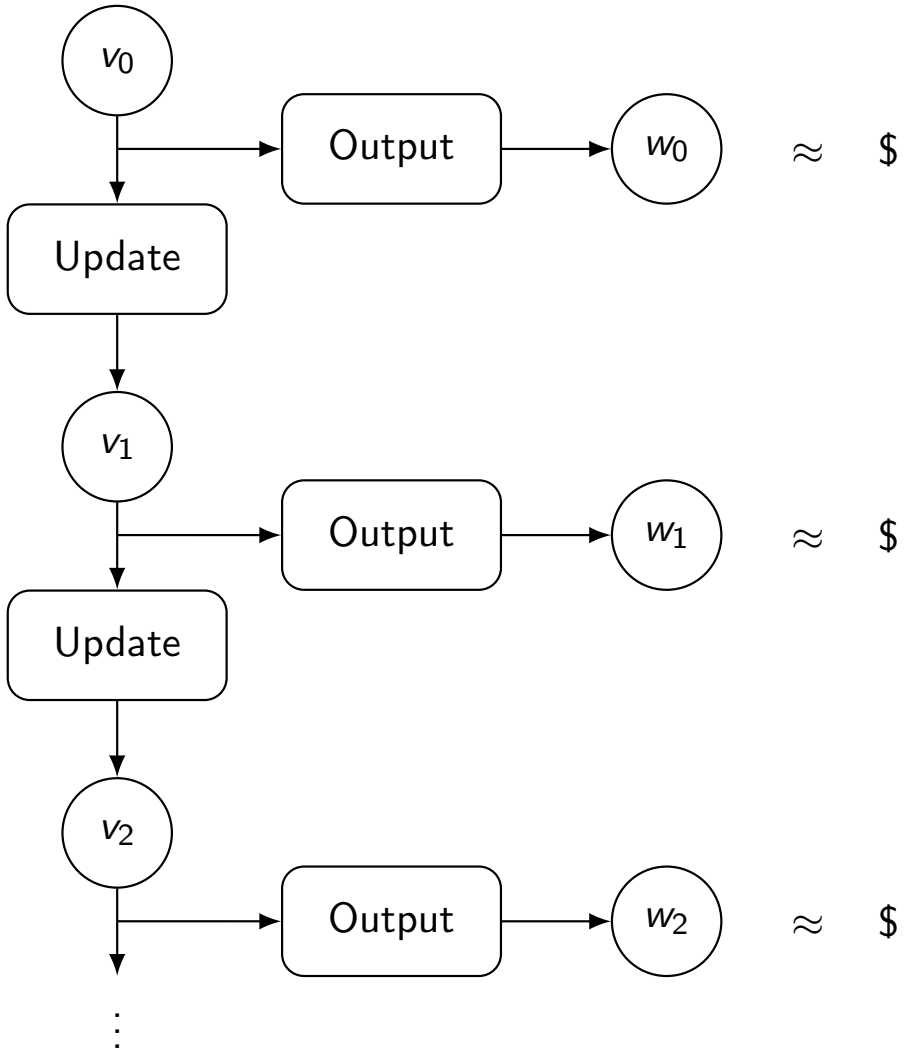
Pseudorandom Generator (PRG)



Pseudorandom Generator (PRG)



Pseudorandom Generator (PRG)



Weak Pseudorandom Generator

Linear congruential generator (LCG)

Parameters $M \in \mathbb{N}$, $a \in \mathbb{Z}_M^*$, $b \in \mathbb{Z}_M$

Seed $v_0 \xleftarrow{\$} \mathbb{Z}_M$

Update $v_{i+1} = a \cdot v_i + b \bmod M$

Output $w_i = k$ most significant bits (MSB) of v_i

Weak Pseudorandom Generator

Linear congruential generator (LCG)

Parameters	$M \in \mathbb{N}$, $a \in \mathbb{Z}_M^*$, $b \in \mathbb{Z}_M$
Seed	$v_0 \xleftarrow{\$} \mathbb{Z}_M$
Update	$v_{i+1} = a \cdot v_i + b \bmod M$
Output	$w_i = k$ most significant bits (MSB) of v_i

After seeing some outputs:

- can recover v_0, when not truncated ($k > \log_2 M$)

Weak Pseudorandom Generator

Linear congruential generator (LCG)

Parameters $M \in \mathbb{N}$, $a \in \mathbb{Z}_M^*$, $b \in \mathbb{Z}_M$

Seed $v_0 \xleftarrow{\$} \mathbb{Z}_M$

Update $v_{i+1} = a \cdot v_i + b \bmod M$

Output $w_i = k$ most significant bits (MSB) of v_i

After seeing some outputs:

- can recover v_0, when not truncated ($k > \log_2 M$)
- can predict outputs
  even when $k = 1$ and a, b secret [Ste87]

Weak Pseudorandom Generator

LCG secure when used in protocols?

Is an LCG secure when used in protocols?

Weak Pseudorandom Generator

LCG secure when used in protocols?

Is an LCG secure when used in protocols?

It depends:

- Yes! for randomness for ElGamal [Kos02]

Weak Pseudorandom Generator

LCG secure when used in protocols?

Is an LCG secure when used in protocols?

It depends:

- Yes! for randomness for ElGamal [Kos02]
- No! for DSA [BGM97]

Weak Pseudorandom Generator

LCG secure when used in protocols?

Is an LCG secure when used in protocols?

It depends:

- Yes! for randomness for ElGamal [Kos02]
- No! for DSA [BGM97]
- Time/memory tradeoff: with RSA key generation [FTZ13]

Weak Pseudorandom Generator

LCG secure when used in protocols?

Is an LCG secure when used in protocols?

It depends:

- Yes! for randomness for ElGamal [Kos02]
- No! for DSA [BGM97]
- Time/memory tradeoff: with RSA key generation [FTZ13]

- Open: PKCS#1 v1.5 encryption

Weak Pseudorandom Generator

LCG secure when used in protocols?

Is an LCG secure when used in protocols?

It depends:

- Yes! for randomness for ElGamal [Kos02]
- No! for DSA [BGM97]
- Time/memory tradeoff: with RSA key generation [FTZ13]
  - → this paper: polynomial time attack in $\log N$
- Open: PKCS#1 v1.5 encryption
  - → this paper: polynomial time attack in $\log N$

Weak Pseudorandom Generator

Algebraic generator

Parameters	$M \in \mathbb{N}$, polynomial F over $\mathbb{Z}_M$
Seed	$v_0 \xleftarrow{\$} \mathbb{Z}_M$
Update	$v_{i+1} = F(v_i) \bmod M$
Output	$w_i = k$ most significant bits of v_i

Weak Pseudorandom Generator

Algebraic generator

Parameters $M \in \mathbb{N}$, polynomial F over $\mathbb{Z}_M$

Seed $v_0 \xleftarrow{\$} \mathbb{Z}_M$

Update $v_{i+1} = F(v_i) \bmod M$

Output $w_i = k$ most significant bits of v_i

Is it secure?

Weak Pseudorandom Generator

Algebraic generator

Parameters	$M \in \mathbb{N}$, polynomial F over $\mathbb{Z}_M$
Seed	$v_0 \xleftarrow{\$} \mathbb{Z}_M$
Update	$v_{i+1} = F(v_i) \bmod M$
Output	$w_i = k$ most significant bits of v_i

Is it secure?

No! [BVZ12]

Weak Pseudorandom Generator

Algebraic generator

Parameters	$M \in \mathbb{N}$, polynomial F over $\mathbb{Z}_M$
Seed	$v_0 \xleftarrow{\$} \mathbb{Z}_M$
Update	$v_{i+1} = F(v_i) \bmod M$
Output	$w_i = k$ most significant bits of v_i

Is it secure?

No! [BVZ12] but complex analysis and no complexity...

Introduction
Contributions

- Toolbox for *Coppersmith methods*:
  - Check success condition;
  - Compute complexity (size of the lattice);
  - Use *analytic combinatorics*;

Introduction
Contributions

- Toolbox for *Coppersmith methods*:
  - Check success condition;
  - Compute complexity (size of the lattice);
  - Use *analytic combinatorics*;
- Applications to cryptanalysis of *weak PRG*:
  - Algebraic PRG
    - Compute complexity of [BVZ12];
  - RSA key generation N with LCG
    - Polynomial time (in $\log N$) attacks (instead of exponential [FTZ13]);
  - PKCS#1 v1.5 padding with LCG
    - Polynomial time attacks (in $\log N$).

Analytic Combinatorics

Overview

- Introduced by Flajolet and Sedgewick [FS09]
- Goal: count combinatorial objects
- Two steps:

Analytic Combinatorics

Overview

- Introduced by Flajolet and Sedgewick [FS09]
- Goal: count combinatorial objects
- Two steps:
  1. Compute generating function (=formal series $P(z)$)

Analytic Combinatorics

Overview

- Introduced by Flajolet and Sedgewick [FS09]
- Goal: count combinatorial objects
- Two steps:
  1. Compute generating function (=formal series $P(z)$)
  2. Do the counting from $P(z)$:
     exact: use any CAS (Taylor expansion)
     asymptotics: easy using the "transfer theorem"

Analytic Combinatorics

Overview

- Introduced by Flajolet and Sedgewick [FS09]
- Goal: count combinatorial objects
- Two steps:
  1. Compute generating function (=formal series $P(z)$)
     $\longrightarrow$ manual but easy step (in our case) using a "dictionary"
  2. Do the counting from $P(z)$:
     exact: use any CAS (Taylor expansion)
     asymptotics: easy using the "transfer theorem"

Example of Use of Analytic Combinatorics

Number n_d of polynomials of total degree d of the form:

$$x^i \cdot (y^4 + 1)^j \qquad \text{for any } i \geq 1 \text{ and } j \geq 0$$

Example of Use of Analytic Combinatorics

Number n_d of polynomials of total degree d of the form:

$$x^i \cdot (y^4 + 1)^j \qquad \text{for any } i \geq 1 \text{ and } j \geq 0$$

1. Generating function:

$$P(z) = \sum_{d=0}^{\infty} n_d z^d = \frac{z}{1-z} \cdot \frac{1}{1-z^4}$$

Example of Use of Analytic Combinatorics

Number n_d of polynomials of total degree d of the form:

$$x^i \cdot (y^4 + 1)^j \qquad \text{for any } i \geq 1 \text{ and } j \geq 0$$

1. Generating function:

$$P(z) = \sum_{d=0}^{\infty} n_d z^d = \frac{z}{1-z} \cdot \frac{1}{1-z^4}$$

2. Count:

Example of Use of Analytic Combinatorics

Number n_d of polynomials of total degree d of the form:

$$x^i \cdot (y^4 + 1)^j \qquad \text{for any } i \geq 1 \text{ and } j \geq 0$$

1. Generating function:

$$P(z) = \sum_{d=0}^{\infty} n_d z^d = \frac{z}{1-z} \cdot \frac{1}{1-z^4}$$

2. Count:

   exact: use any CAS (Taylor expansion)

Example of Use of Analytic Combinatorics

Number n_d of polynomials of total degree d of the form:

$$x^i \cdot (y^4 + 1)^j \qquad \text{for any } i \geq 1 \text{ and } j \geq 0$$

1. Generating function:

$$P(z) = \sum_{d=0}^{\infty} n_d z^d = \frac{z}{1-z} \cdot \frac{1}{1-z^4}$$

2. Count:

exact: use any CAS (Taylor expansion)

asymptotics: use transfer theorem

$$P(z) \underset{z \to 1}{\sim} \frac{1}{4 \cdot (1-z)^2} \qquad \Longrightarrow \qquad n_d \underset{d \to \infty}{\sim} \frac{d}{4}$$

Attack on Algebraic PRG (I)

Parameters $M \in \mathbb{N}$, polynomial F of degree d over $\mathbb{Z}_M$

Seed $v_0 \xleftarrow{\$} \mathbb{Z}_M$

Update $v_{i+1} = F(v_i) \bmod M$

Output $w_i = k$ most significant bits of v_i

Attack on Algebraic PRG (I)

Parameters $M \in \mathbb{N}$, polynomial F of degree d over $\mathbb{Z}_M$

Seed $v_0 \xleftarrow{\$} \mathbb{Z}_M$

Update $v_{i+1} = F(v_i) \bmod M$

Output $w_i = k$ most significant bits of v_i

We know w_i and we write

$$v_i = w_i \cdot 2^{n-k} + x_i \quad \text{for } 0 \leq i \leq n+1$$

We want to solve:

$$\begin{cases} f_0(x_0, x_1) = w_1 \cdot 2^{n-k} + x_1 - F(w_0 \cdot 2^{n-k} + x_0) \equiv 0 \mod M \\ \vdots \\ f_n(x_n, x_{n+1}) = w_{n+1} \cdot 2^{n-k} + x_{n+1} - F(w_n \cdot 2^{n-k} + x_n) \equiv 0 \mod M \end{cases}$$

Attack on Algebraic PRG (II)

We construct the polynomials of degree at most dm:

$$x_0{}^j \cdot f_0^{i_0} \cdots f_n^{i_n}$$

Attack on Algebraic PRG (II)

We construct the polynomials of degree at most dm:

$$x_0{}^j \cdot f_0^{i_0} \cdots f_n^{i_n}$$

Using our toolbox, the attack works when $m \to \infty$:

$$\frac{\lceil \log_2 M \rceil - k}{\lceil \log_2 M \rceil} \leq \frac{d^{n+1} - 1}{d^{n+2} - 1} \approx \frac{1}{d}$$

RSA Key Generation with LCG (I)

Factorize an RSA modulus $N = pq$ where

$$p = v_0 + Mv_1 + \cdots + M^n v_n \quad \text{and} \quad q = w_0 + Mw_1 + \cdots + M^n w_n$$

and $v_0, \ldots, v_n$ and $w_0, \ldots, w_n$ are non-truncated output of a LCG.

RSA Key Generation with LCG (I)

Factorize an RSA modulus $N = pq$ where

$$p = v_0 + Mv_1 + \cdots + M^n v_n \quad \text{and} \quad q = w_0 + Mw_1 + \cdots + M^n w_n$$

and $v_0, \ldots, v_n$ and $w_0, \ldots, w_n$ are non-truncated output of a LCG.

We have:

$$\begin{cases} f = (v_0 + \cdots + M^n v_n)(w_0 + \cdots + M^n w_n) \equiv 0 \bmod N \\ g_0 = v_1 - (av_0 + b) \equiv 0 \bmod M \\ \vdots \\ g_{n-1} = v_n - (av_{n-1} + b) \equiv 0 \bmod M \\ h_0 = w_1 - (aw_0 + b) \equiv 0 \bmod M \\ \vdots \\ h_{n-1} = w_n - (aw_{n-1} + b) \equiv 0 \bmod M \end{cases}$$

RSA Key Generation with LCG (II)

We construct the polynomials of degree $2t$:

$$\tilde{f}_{i_0,\ldots,i_n,j_0,\ldots,j_n,k} = {v_0}^{i_0} \ldots {v_n}^{i_n} \cdot {w_0}^{j_0} \ldots {w_n}^{j_n} \cdot f^k \bmod N^k$$
$$\text{with } 1 \leq k < t,\ (i_0 = 0 \text{ or } j_0 = 0)$$
$$\text{and } \deg(\tilde{f}_{\ldots}) = i_0 + \cdots + i_n + j_0 + \cdots + j_n + 2k < 2t$$
$$\tilde{g}_{i_0,\ldots,i_n,j_0,\ldots,j_n} = g_0^{i_0} \ldots g_{n-1}^{i_{n-1}} \cdot {v_n}^{i_n} \cdot h_0^{j_0} \ldots h_n^{j_{n-1}} \cdot {w_n}^{j_n} \bmod M^\ell$$
$$\text{with } 1 \leq \ell = i_0 + \cdots + i_{n-1} + j_0 + \cdots + j_{n-1}$$
$$\text{and } \deg(\tilde{g}_{\ldots}) = i_0 + \cdots + i_n + j_0 + \cdots + j_n < 2t \ .$$

RSA Key Generation with LCG (II)

We construct the polynomials of degree $2t$:

$$\tilde{f}_{i_0,\dots,i_n,j_0,\dots,j_n,k} = {v_0}^{i_0} \dots {v_n}^{i_n} \cdot {w_0}^{j_0} \dots {w_n}^{j_n} \cdot f^k \bmod N^k$$
$$\text{with } 1 \leq k < t,\ (i_0 = 0 \text{ or } j_0 = 0)$$
$$\text{and } \deg(\tilde{f}_{\dots}) = i_0 + \dots + i_n + j_0 + \dots + j_n + 2k < 2t$$
$$\tilde{g}_{i_0,\dots,i_n,j_0,\dots,j_n} = g_0^{i_0} \dots g_{n-1}^{i_{n-1}} \cdot {v_n}^{i_n} \cdot h_0^{j_0} \dots h_n^{j_{n-1}} \cdot {w_n}^{j_n} \bmod M^\ell$$
$$\text{with } 1 \leq \ell = i_0 + \dots + i_{n-1} + j_0 + \dots + j_{n-1}$$
$$\text{and } \deg(\tilde{g}_{\dots}) = i_0 + \dots + i_n + j_0 + \dots + j_n < 2t \ .$$

Using our toolbox:
$\forall n \geq 2$, $\exists t$, attack works in time poly in $\log N$.

PKCS#1 v1.5 = padding for RSA: encryption of m:

$$c = \mu(m, r)^e \bmod N \quad \text{and} \quad \mu(m, r) = 0002_{16}||r||00_{16}||m$$

with r random bit string.

Attack similar to the previous one,
when r generated by LCG modulo M.
Work for $M < N^{1/e}$.

Thank you for your attention!
Questions?

- Toolbox for *Coppersmith methods*:
  - Check success condition;
  - Compute complexity (size of the lattice);
  - Use *analytic combinatorics*;
- Applications to cryptanalysis of *weak PRG*:
  - Algebraic PRG
    - Compute complexity of [BVZ12];
  - RSA key generation N with LCG
    - Polynomial time (in $\log N$) attacks (instead of exponential [FTZ13]);
  - PKCS#1 v1.5 padding with LCG
    - Polynomial time attacks (in $\log N$).

References I

Mihir Bellare, Shafi Goldwasser, and Daniele Micciancio.

"pseudo-random" number generation within cryptographic algorithms: The DDS case.

In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 277–291. Springer, August 1997.

Johannes Blömer and Alexander May.

A tool kit for finding small roots of bivariate polynomials over the integers.

In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 251–267. Springer, May 2005.

References II

Aurélie Bauer, Damien Vergnaud, and Jean-Christophe Zapalowicz.

Inferring sequences produced by nonlinear pseudorandom number generators using Coppersmith's methods.

In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 609–626. Springer, May 2012.

Don Coppersmith.

Finding a small root of a bivariate integer equation; factoring with high bits known.

In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 178–189. Springer, May 1996.

References III

Don Coppersmith.

Finding a small root of a univariate modular equation.

In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 155–165. Springer, May 1996.

Philippe Flajolet and Robert Sedgewick.

Analytic Combinatorics.

Cambridge University Press, January 2009.

Pierre-Alain Fouque, Mehdi Tibouchi, and Jean-Christophe Zapalowicz.

Recovering private keys generated with weak PRNGs.

In Martijn Stam, editor, *14th IMA International Conference on Cryptography and Coding*, volume 8308 of *LNCS*, pages 158–172. Springer, December 2013.

References IV

Nick Howgrave-Graham.

Finding small roots of univariate modular equations revisited.

In Michael Darnell, editor, *6th IMA International Conference on Cryptography and Coding*, volume 1355 of *LNCS*, pages 131–142. Springer, December 1997.

Ellen Jochemsz and Alexander May.

A strategy for finding roots of multivariate polynomials with new applications in attacking RSA variants.

In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 267–282. Springer, December 2006.

Takeshi Koshiba.

On sufficient randomness for secure public-key cryptosystems.

In David Naccache and Pascal Paillier, editors, *PKC 2002*, volume 2274 of *LNCS*, pages 34–47. Springer, February 2002.

References V

Alexander May and Maike Ritzenhofen.

Solving systems of modular equations in one variable: How many RSA-encrypted messages does eve need to know?

In Ronald Cramer, editor, *PKC 2008*, volume 4939 of *LNCS*, pages 37–46. Springer, March 2008.

Alexander May and Maike Ritzenhofen.

Implicit factoring: On polynomial time factoring given only an implicit hint.

In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 1–14. Springer, March 2009.

Maike Ritzenhofen.

On efficiently calculating small solutions of systems of polynomial equations: lattice-based methods and applications to cryptography.

PhD thesis, Ruhr University Bochum, 2010.

References VI

Jacques Stern.

Secret linear congruential generators are not cryptographically secure.

In *28th FOCS*, pages 421–426. IEEE Computer Society Press, October 1987.