Recovering Private Keys Generated With Weak PRNGs

Pierre-Alain Fouque (Univ. Rennes 1)

Mehdi Tibouchi (NTT Secure Platform Lab.)

Jean-Christophe Zapalowicz (Inria)

Journées C2 2014

About this talk

"Generating secure public keys in the real world is challenging"
(Lenstra et al. 2012)
"We discovered that insecure RNGs are in widespread use"
(Heninger et al. 2012)

About this talk

"Generating secure public keys in the real world is challenging"
(Lenstra et al. 2012)
"We discovered that insecure RNGs are in widespread use"
(Heninger et al. 2012)

A public-key scheme based on the hardness of

- the discrete logarithm problem
- the factorization

Known values:

- the public key
- a Linear Congruential Generator (LCG) used to generate the secret key

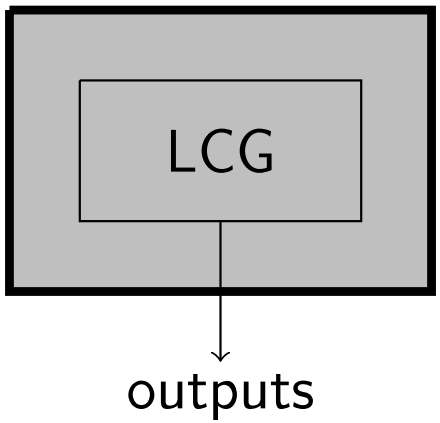
Question: Is it secure?

Related Work: Bellare et al. (1997) show that using a LCG for computing the nonces in DSA does seriously break security

Linear Congruential Generator is weak

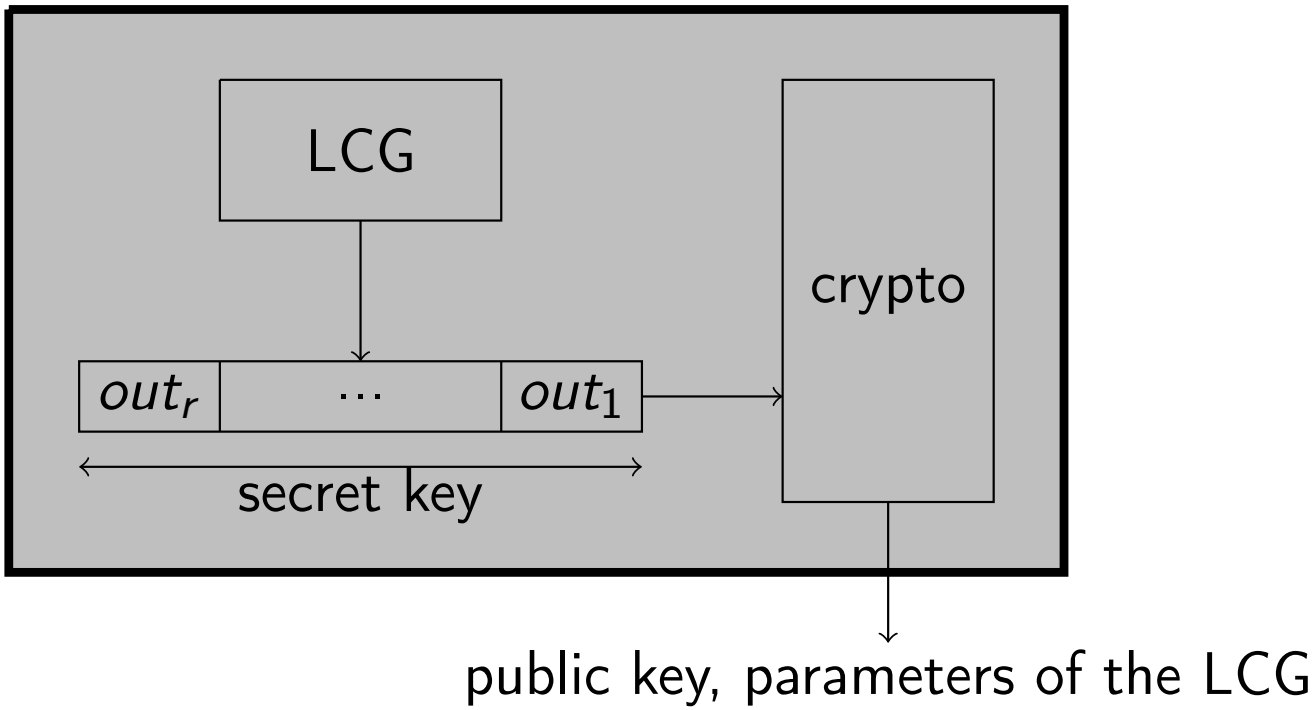
Heavily studied: Stern (1987), Frieze et al. (1988), Boyar (1989), Joux and Stern (1998), Contini and Shparlinski (2005)

Advantages: efficient, very small footprint, easy to implement, good statistical properties...
Drawback: cryptographically insecure:
$\Rightarrow$ from some outputs, possibility to recover the seed and the parameters



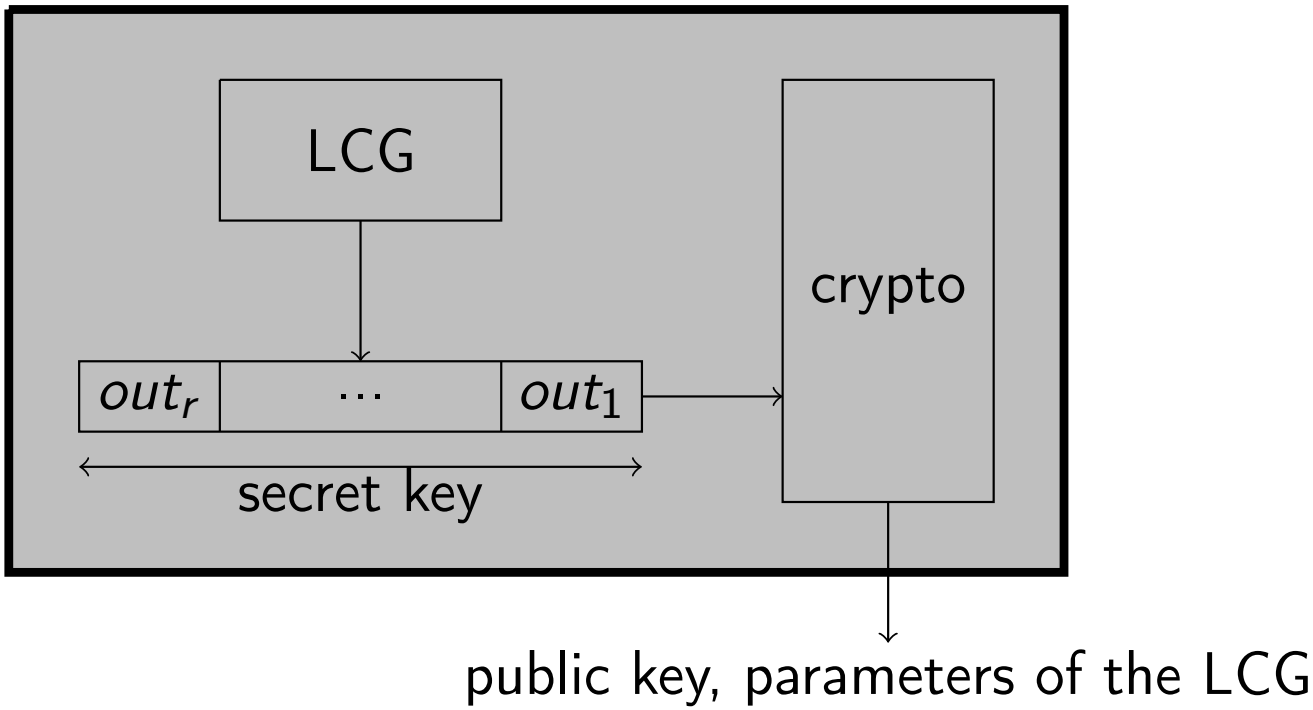
Is this scheme weak?

Reality: no access to the outputs of the LCG



Is this scheme weak?

Reality: no access to the outputs of the LCG

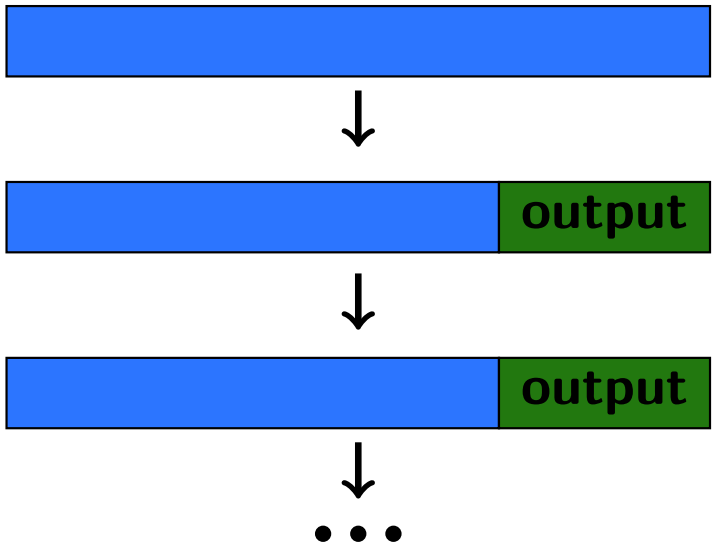


Cryptanalysis: search of the m-bit seed of the LCG

- Exhaustive search: complexity in $O(2^m)$
- **Our work:**
  Time/Memory tradeoff: complexity between $O(2^{m/2})$ and $O(2^m)$

Linear Congruential Generator

M: known integer
$a, b < M$: known integers
$s < M$: secret seed



seed s

$v_1 = as + b \bmod M$

$v_2 = av_1 + b \bmod M$

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

The LCG outputs the k least (or most) significant bits o_i of v_i

Overview of our cryptanalysis (1)

$v_{i+1} = a \cdot v_i + b \bmod M$
$\Rightarrow v_i = a^i \cdot s + b \cdot (1 + a + \cdots + a^{i-1}) = a^i \cdot s + b_i \bmod M$
constants b_i publicly computable $\rightarrow b = 0$

$$v_i = a^i \cdot s \bmod M$$

$$\text{secret key } x = o_0 + 2^k o_1 + \cdots + 2^{(r-1)k} o_{r-1}$$

$$\text{with } o_i = \text{lsb}_k(v_i)$$

Overview of our cryptanalysis (1)

$v_{i+1} = a \cdot v_i + b \bmod M$
$\Rightarrow v_i = a^i \cdot s + b \cdot (1 + a + \cdots + a^{i-1}) = a^i \cdot s + b_i \bmod M$
constants b_i publicly computable $\rightarrow b = 0$

$$v_i = a^i \cdot s \bmod M$$

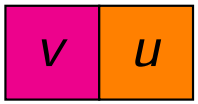
$$\text{secret key } x = o_0 + 2^k o_1 + \cdots + 2^{(r-1)k} o_{r-1}$$

$$\text{with } o_i = \text{lsb}_k(v_i)$$

Main idea of the attack:

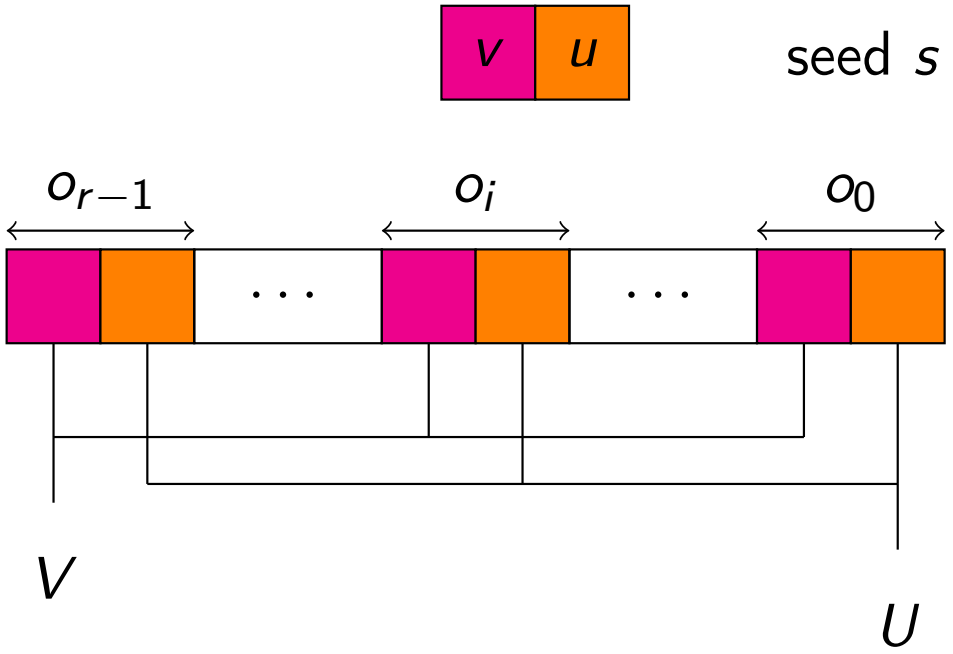
- split the m-bit seed into two part $s = u + 2^{m/2} v$
- write the secret key as $x = U + V$
  - U depends only on the $\frac{m}{2}$-bit value u
  - V depends only on the $\frac{m}{2}$-bit value v
- take carries and overflows into account

Overview of our cryptanalysis (2):

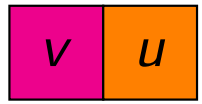


seed s

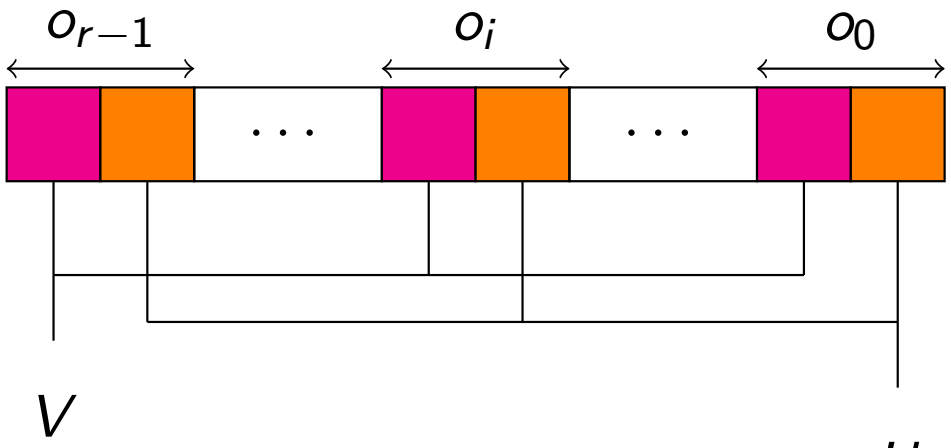
Overview of our cryptanalysis (2):



Overview of our cryptanalysis (2):

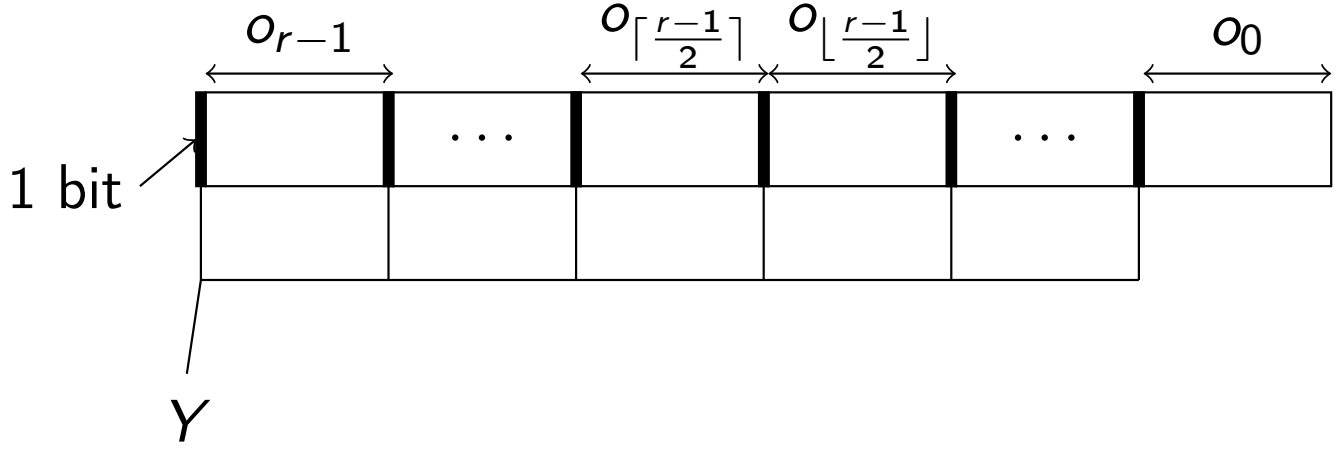


seed s



secret key x

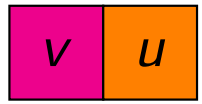
$$o_i = a^i \cdot s \bmod 2^k = a^i \cdot u \bmod 2^k + 2^{k/2}(a^i \cdot v \bmod 2^{k/2}) - \epsilon_i \cdot 2^k$$



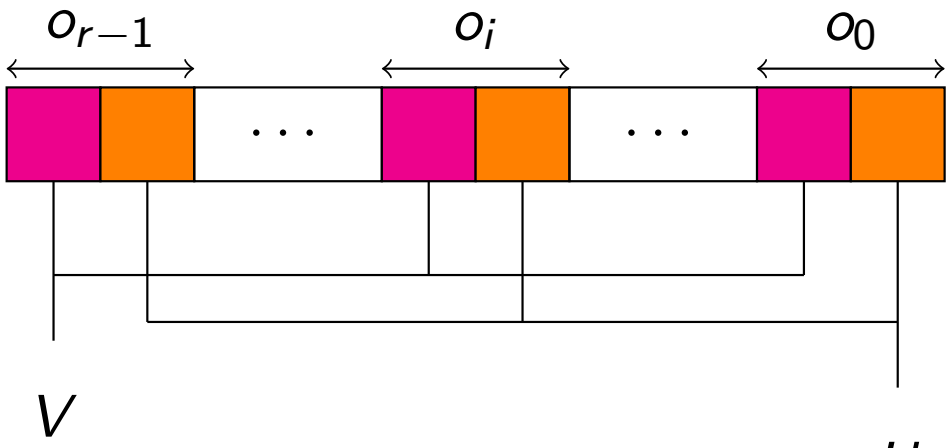
carries

$(M = 2^k)$

Overview of our cryptanalysis (2):

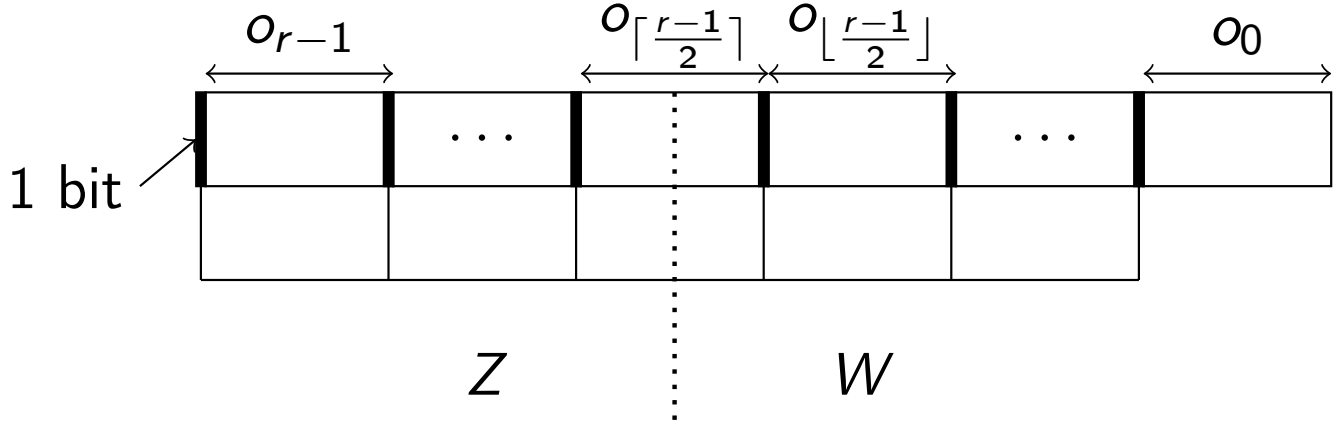


seed s



secret key x

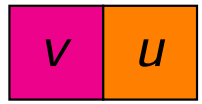
$$o_i = a^i \cdot s \bmod 2^k = a^i \cdot u \bmod 2^k + 2^{k/2}(a^i \cdot v \bmod 2^{k/2}) - \epsilon_i \cdot 2^k$$



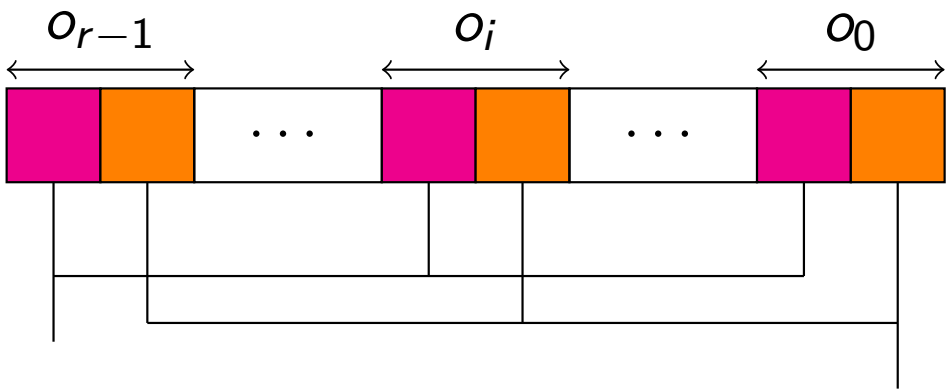
carries

$(M = 2^k)$

Overview of our cryptanalysis (2):

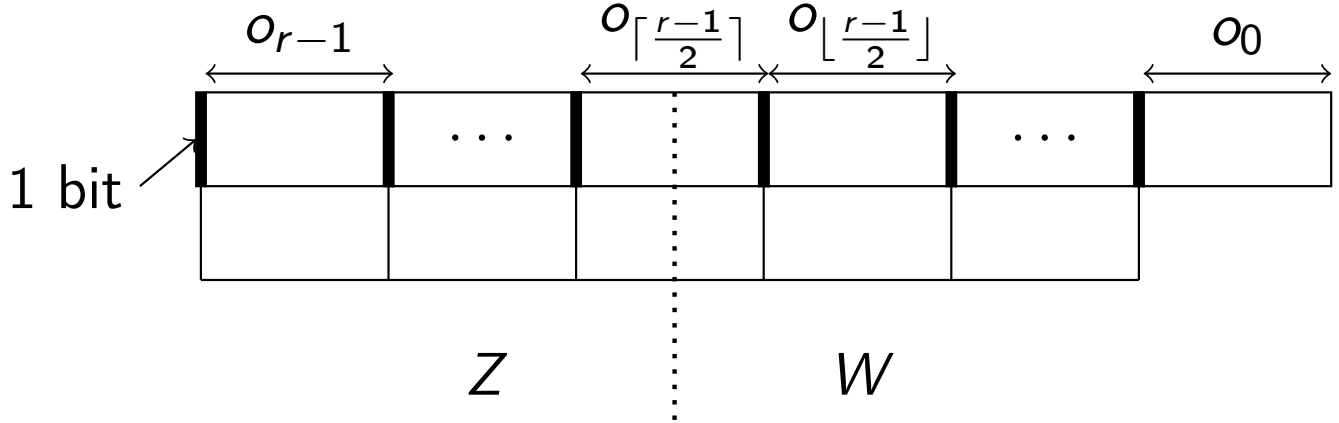


seed s



secret key x

$$\mathbf{x = U + V - W - Z}$$



carries

$(M = 2^k)$

Results on the discrete logarithm case: Introduction

Discrete logarithm problem:

- cyclic group $\mathbb{G}$ of prime order q and generator g
  $\rightarrow (\mathbb{G}, q, g)$ public parameters
- secret key: $x \in \mathbb{Z}_q$
- public key: $h = g^x$

$\Rightarrow$Hard to retrieve x from h

$$x = o_0 + 2^k o_1 + \cdots + 2^{(r-1)k} o_{r-1}$$

We consider the following cases:

- use of a non-truncated LCG: $M = 2^k$
- use of a truncated LCG: $M > 2^k$ (see paper)

Discrete logarithm case with a non-truncated LCG (1)

$$x = U + V - W - Z$$

where:

$$U = \sum_{i=0}^{r-1} 2^{ik} \cdot (a^i u \bmod 2^k)$$

$$V = \sum_{i=0}^{r-1} 2^{ik+k/2} \cdot (a^i v \bmod 2^{k/2})$$

$$Y = \sum_{i=0}^{r-1} 2^{(i+1)k} \cdot \varepsilon_i = \underbrace{\sum_{i=0}^{\lfloor \frac{r-1}{2} \rfloor} 2^{(i+1)k} \cdot \varepsilon_i}_{W} + \underbrace{\sum_{i=\lceil \frac{r-1}{2} \rceil}^{r-1} 2^{(i+1)k} \cdot \varepsilon_i}_{Z} \qquad \varepsilon_i \in \{0,1\}$$

$$\boxed{g^{U-Z} = h \cdot g^{W-V}}$$

Discrete logarithm case with a non-truncated LCG (2)

$$\boxed{g^{U-Z} = h \cdot g^{W-V}}$$

Collision search

- U (resp. V) depends only on the $\frac{k}{2}$-bit value u (resp. v)
  $\Rightarrow 2^{k/2}$ possible values U_i and $2^{k/2}$ possible values V_s
- W and Z take carries and overflows into account
  $\Rightarrow 2^{r/2}$ possible values W_t and $2^{r/2}$ possible values Z_j

(k: size of the seed, r: number of used outputs)

Attack:

- Compute the hash table H by storing i, j at $H(g^{U_i - Z_j})$
  $\Rightarrow O(2^{\frac{k+r}{2}})$ in memory
- For each (V_s, W_t) do:
  $\Rightarrow O(2^{\frac{k+r}{2}})$ in time
  - if $H(h \cdot g^{W_t - V_s})$ exists then return $x \leftarrow U_i + V_s - Z_j - W_t$

Concretly

Secret size	Modulus	Attack complexity
160	2^{32}	$2^{18.5}$
160	2^{64}	$2^{33.5}$
256	2^{32}	2^{20}
256	2^{64}	2^{34}
512	2^{32}	2^{24}
512	2^{64}	2^{36}
1024	2^{64}	2^{40}
2048	2^{64}	2^{48}

Results on the factorization case: Introduction

Factorization problem:

p, q two large primes

Given $N = pq$, retrieve p and q

$$p = o_0 + 2^k o_1 + \cdots + 2^{(r-1)k} o_{r-1}$$

We consider the following cases:

- basic prime generation
  - use of a non-truncated LCG: $M = 2^k$
  - use of a truncated LCG: $M > 2^k$ (see paper)
- PRIMEINC method (see paper)
  - use of a non-truncated LCG: $M = 2^k$
  - use of a truncated LCG: $M > 2^k$

Factorization case with a non-truncated LCG (1)

$$p = \underbrace{U - Z}_{A} + \underbrace{V - W}_{B}$$

Point of view of an attacker:

- $2^{\frac{k+r}{2}}$ possible values $A_{i,j}$
- $2^{\frac{k+r}{2}}$ possible values $B_{s,t}$

How to find the correct value A and B?

$$\text{if } q \not| c, \text{ then } \boxed{\gcd(c(A+B), N) = p}$$

Use of the Multipoint evaluation of univariate polynomials:

- $P(x) \in \mathbb{Z}_N[x]$ a polynomial of degree d
- Evaluate P at d distinct points
- Quasi-linear complexity $\widetilde{O}(d)$ using divide-and-conquer approach

Factorization case with a non-truncated LCG (2)

Attack:

- Compute the polynomial of degree $2^{\frac{k+r}{2}}$:

$$P(X) = \prod_{s,t}(X + B_{s,t}) \bmod N$$

  $\Rightarrow \widetilde{O}(2^{\frac{k+r}{2}})$ in memory
- Multi-evaluate P at the points $A_{i,j}$

  $\Rightarrow \widetilde{O}(2^{\frac{k+r}{2}})$ in time
- For each point $A_{i,j}$ do:
  - if $\gcd(P(A_{i,j}), N) \neq 1$ then return $\gcd(P(A_{i,j}), N)$

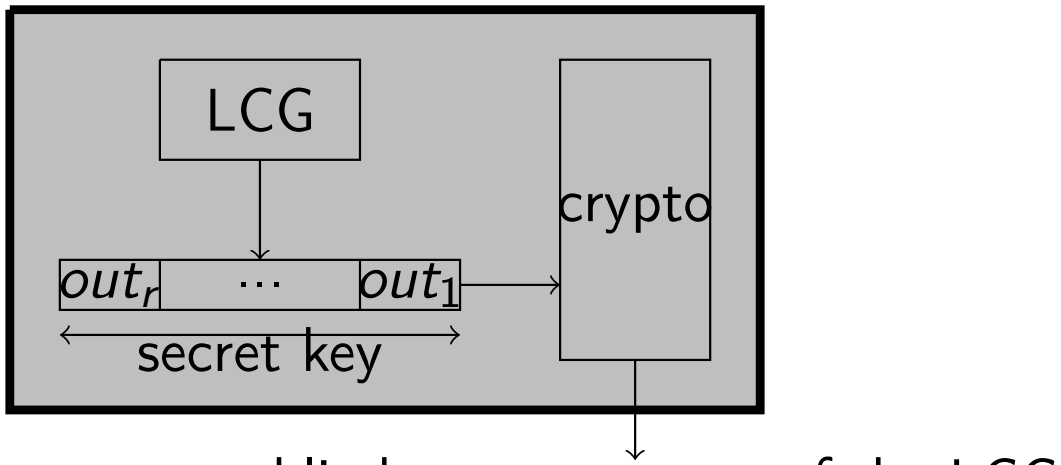
For the correct value A:

$$P(A) = c(A + B)$$

with $c = \prod_{s,t \mid B_{s,t} \neq B}(A + B_{s,t})$

Conclusion

Discrete logarithm-based and factorization-based schemes treated



Exhaustive search: complexity in $O(2^m)$

Our Time/Memory tradeoff: complexity between $O(2^{m/2})$ and $O(2^m)$

Open Questions:

- What if the parameters of the LCG are unknown
- Generalization to other PRNG

Thank you for your attention