Kleptography
An overview

Guglielmo Morgari

Telsy SpA and Politecnico di Torino

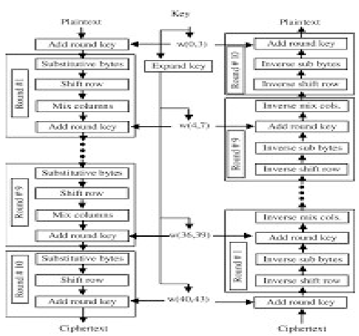
19 January 2021

Talk structure

- Introduction and scenario
- Definition of kleptography
- Kleptographic RSA
- Kleptographic Diffie Hellman
- The strange case of EC_DUAL_DRBG
- Conclusions

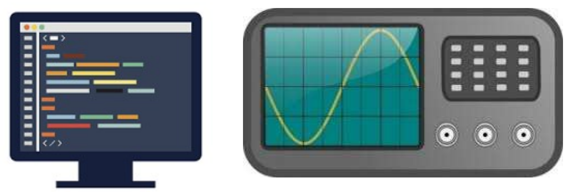
Introduction
Where (in)security come from

Cryptographic primitives (in)security is in the math…



- algebraic attacks
- statistical attacks
- …

… in the (software and hardware) implementation…



- implementation bugs
- side channel attacks
- …

… but it is also a matter of TRUST (in technology providers)

- trivial (but effective) trapdoors
- KLEPTOGRAPHY

Kleptography
Definition

- First works about kleptography and cryptovirology:
  Adam Young, Moti Yung, in the mid-90s

- *Kleptography is the study of stealing information*
  securely *(exclusively) and* ***subliminally*** *(unnoticeably)*

- Asymmetry between reverse engineer and malicious manufacturer
- Black box model (quite realistic)

Cryptographic Device
(**H**ardware **S**ecure **M**odule)
Smart Card
SIM
PC Card
...

- Cryptographic backdoor in public key systems
- Very general problem for cryptographic implementations (mostly hardware) but not only

Kleptography
Esamples

RSA (Young, Yung, 1996)

Diffie Hellman (Young, Yung, 1997)

Dual_EC_DRBG PRNG (NIST SP 800-90A)

RSA Reminder

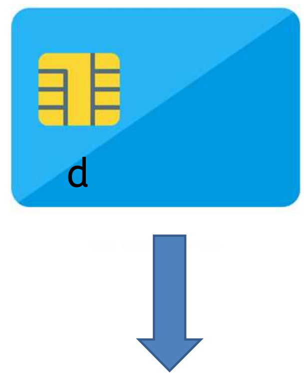
N=pq

N: 2048 bits

p,q: 1024 bits

Public key: (**N**, e)

Private key: d s.t. de=1 mod (p-1)(q-1)

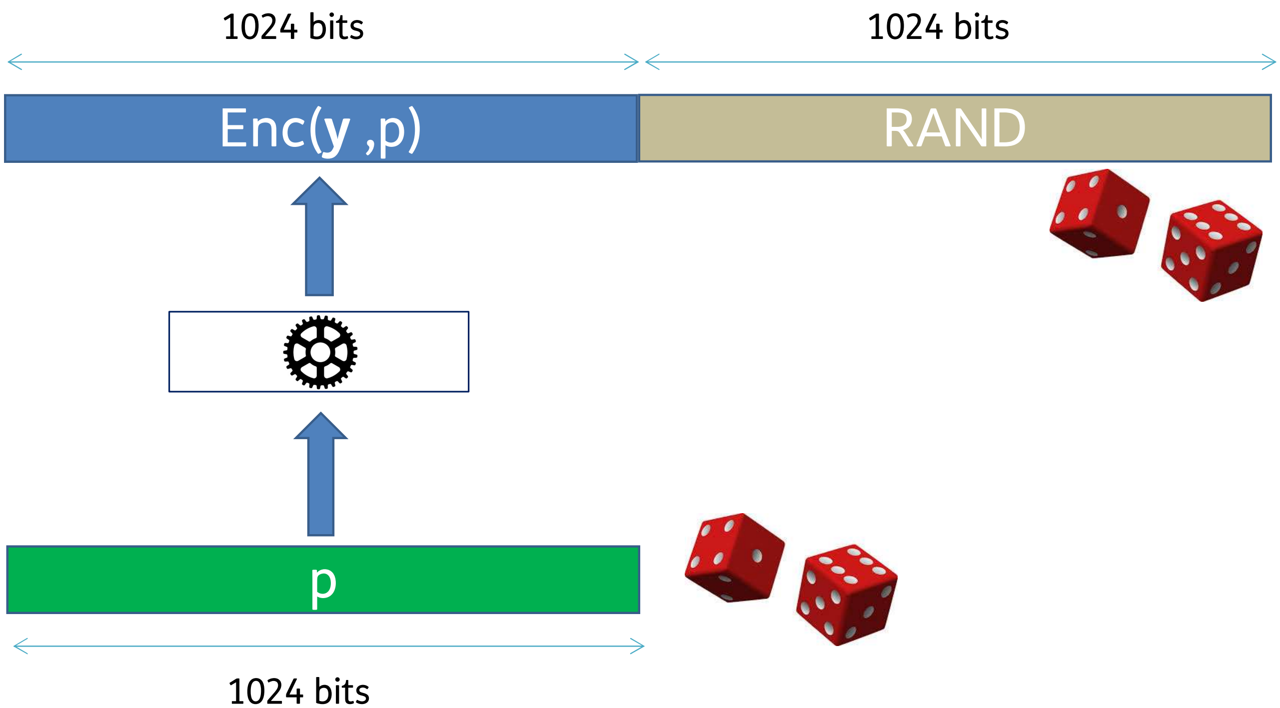


Public key generation inside the device

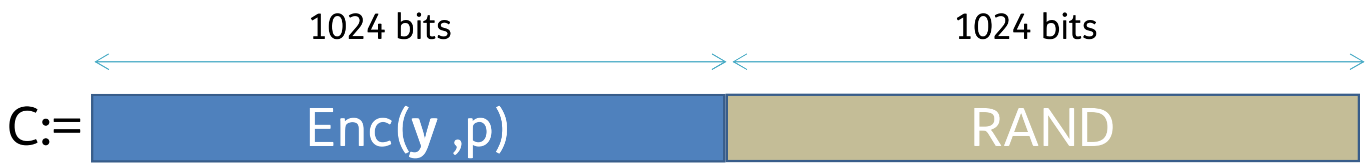
(**N**,e)

security given by the integer factorization problem
knowledge of p or q breaks the system

RSA
Modulus generation



RSA
Attack



C=p*q+r

(q, r) univocally defined

r < p ⇒ size(r) ≤ 1024 bits

If q is not prime:

generate a new p

If q is prime:

N:=p*q, (d,e) computed as usual

N:=C-r

RSA Attack

1024 bits | 1024 bits

C:= Enc(**y**, p) | RAND

N:=C-r

Enc(**y**, p) | RAND-r — if $r \leq RAND$

Enc(**y**, p)-1 | RAND+2^{1024}-r — if $r > RAND$

H_N | L_N — $N = H_N * 2^{1024} + L_N$

p is embedded in H_N

p1:= Dec(**x**, H_N), p2:=Dec(**x**, H_N+1)

if p1|N p:=p1
else (p2|N) p:=p2

The attacker knows **x**, thus given N can recover p

RSA
Observations

- ***subliminally*** *(unnoticeably)*
  output public key (N) is indistinguishable from “normal” public keys

- ***securely*** *(exclusively)*
  M ( )’s private key **x** needed to exploit the trapdoor

- ***forward secrecy***
  If a reverse engineer manages to break the black box (i.e. he finds **y**), he can’t recover user past private keys (d)

Diffie Hellman Reminder

Alice generates a **random** a

Alice sends g^a

Alice computes $K = (g^b)^a$

Bob generates a **random** b

Bob sends g^b

Bob computes $K = (g^a)^b$

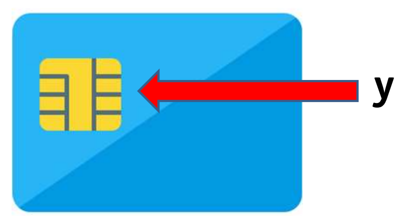
An eavesdropper (Eve)

- knows g^a and g^b but
- is not able to find a or b (**DLP**)
- thus cannot compute K

Diffie Hellman
Kleptographic model

A crypto device is used to implement Diffie Hellman exchange

Secret parameter *a* generation and exponentiations g^a, $(g^b)^a$ made inside the device



The malicious manufacturer (M) owns a Diffie Hellman pair: public key ***y*** and a private key ***x***:

$y=g^x$

The public key *y* is stored inside the device

Diffie Hellman
As it should go

randomly generates a_1		randomly generates b_1
computes $A_1=g^{a1}$		computes $B_1=g^{b1}$
sends A1 and receive B1		sends B1 and receive A1
computes $K1= B1^{a1}$		computes $K1= A1^{b1}$
deletes a_1		deletes b_1
randomly generates a_2		randomly generates b_2
computes $A_2=g^{a2}$		computes $B_2=g^{b2}$
sends A2 and receive B2		sends B2 and receive A2
computes $K2= B2^{a2}$		computes $K2= A2^{b2}$
deletes a_2		deletes b_2
…		…

Diffie Hellman
As it can go  (attack)

randomly generates a_1

computes $A_1=g^{a1}$

sends A1 and receive B1

computes $K1= B1^{a1}$

~~deletes a_1~~ keeps a_1

randomly generates b_1

computes $B_1=g^{b1}$

sends B1 and receive A1

computes $K1= A1^{b1}$

deletes b_1

~~randomly generates~~ set $a_2=H(y^{a1})$ (H hash function)

computes $A_2=g^{a2}$

sends A2 and receive B2

computes $K2= B2^{a2}$

deletes a_2

randomly generates b_2

computes $B_2=g^{b2}$

sends B2 and receive A2

computes $K2= A2^{b2}$

deletes b_2

knows A1 e B2 (seen online)

computes $a_2=H(A1^x)$ ← $A1^x=(g^{a1})^x=(g^x)^{a1}=y^{a1}$

computes $K2= B2^{a2}$

Diffie Hellman
As it can go (attack)

randomly generates a_1	randomly generates b_1
computes $A_1=g^{a1}$	computes $B_1=g^{b1}$
sends A1 and receive B1	sends B1 and receive A1
computes $K1= B1^{a1}$	computes $K1= A1^{b1}$
~~deletes a_1~~ keeps a_1	deletes b_1

~~randomly generates~~ set $a_2=H(y^{a1})$ (H hash function)	randomly generates b_2
computes $A_2=g^{a2}$	computes $B_2=g^{b2}$
sends A2 and receive B2	sends B2 and receive A2
computes $K2= B2^{a2}$	computes $K2= A2^{b2}$
~~deletes a_2~~ deletes a1, keeps a_2	deletes b_2

~~randomly generates~~ set $a_3=H(y^{a2})$	randomly generates b_3
...	...

Diffie Hellman
Observations

- ***subliminally*** *(unnoticeably)*

  output public key (g^a) is indistinguishable from “normal” public keys

- ***securely*** *(exclusively)*

  M ()’s private key **x** needed to exploit the trapdoor

- ***forward secrecy***

  If a reverse engineer manages to break the black box (i.e. he finds **y**), he can’t recover user past private keys (a)

Dual_EC_DRBG
Timeline

2004 Preliminary versions

2006 Published in *NIST SP 800-90*

NIST SP 800-90: Recommendation for Random Number Generation Using Deterministic Random Bit Generators

2012 Maintained in *NIST SP 800-90a*

2015 Withdrawn in *NIST SP 800-90a Rev. 1 in 2015*

Dual_EC_DRBG Parameters

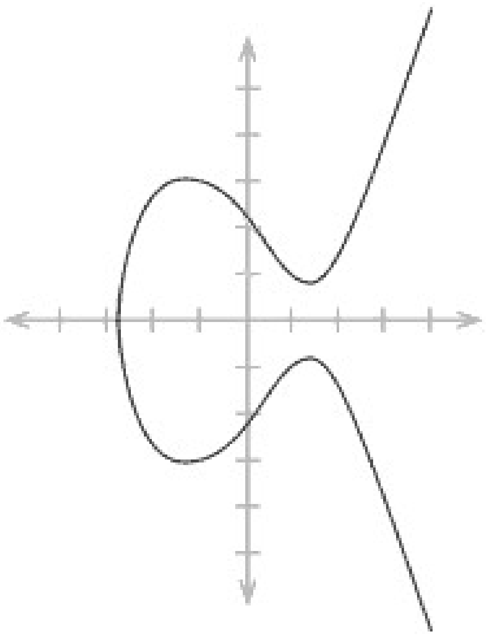
- Defined over three standard elliptic curves:

  P-256, P-384, P-512

- For each curve the following parameters are given:
- $E : y^2 \equiv x^3 + ax + b \pmod{p}$
- **P**: a base point (x_P, y_P)
- **N**: order (P), N prime

Additionally:

- **Q**: a point (x_Q, y_Q) on the curve

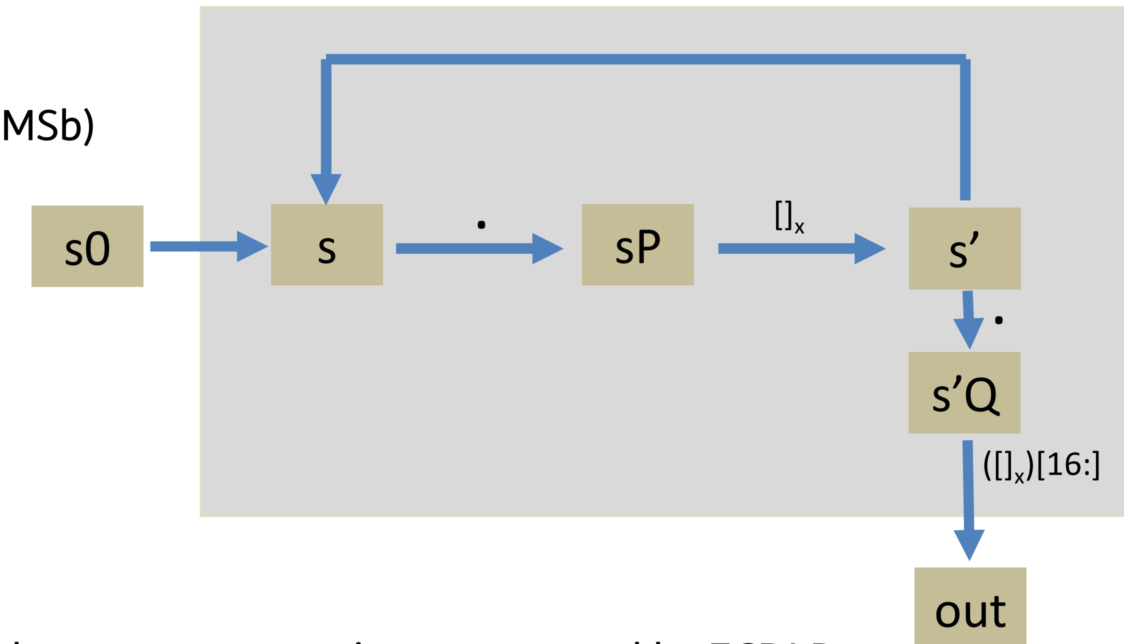


Dual_EC_DRBG Scheme

$[\,]_x$: take x-coordinate

[16:] : take from 17th bit (delete 16 MSb)

• : EC scalar multiplication



- *s0* seed, *s0* < *N*
- *s* < *N*
- *s***P**: internal state

- Both internal state evolution and output computation are secured by ECDLP

Dual_EC_DRBG Attack

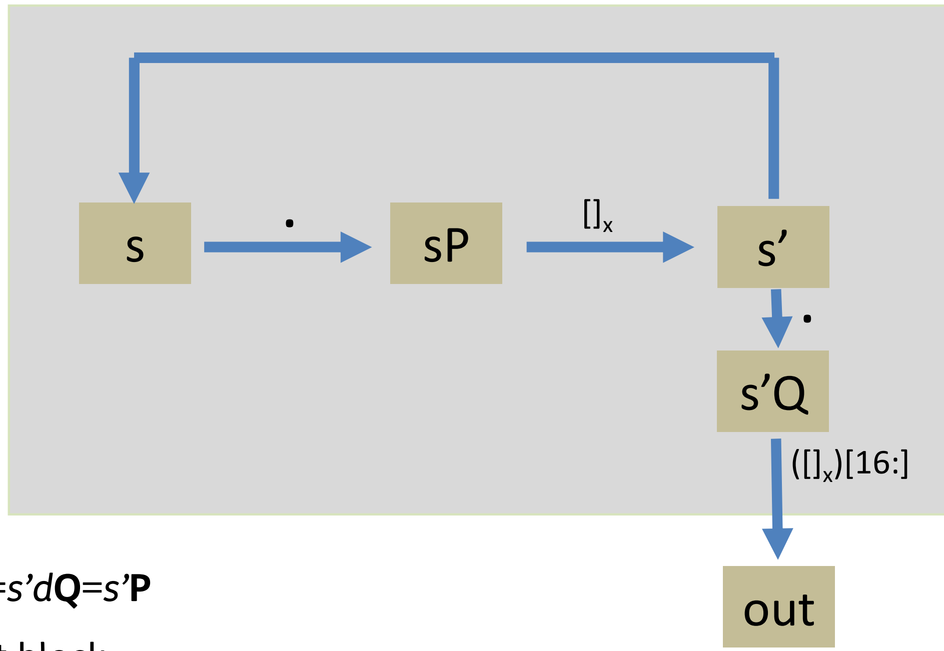
- **P** generator ➡ ∃ *e* / **Q**=*e***P**
- *N* prime ➡ ∃ *d* / *de*=1 mod *N*, **P**=*d***Q**

Attack (**assuming *d* is known**)

- Take **out**
- Compute T={ i | **out**, 0 < i < 2^{16}}
- $\forall t \in T$, define (if possible) $\boldsymbol{Z_t} = (t, \sqrt{t^3 + at + b})$

  Note: ∃t / $\boldsymbol{Z_t}$ = s'**Q**
- $\forall t \in T$, compute $d\boldsymbol{Z_t}$; if $\boldsymbol{Z_t}$ = *s'***Q**, then $d\boldsymbol{Z_t}$=*ds'***Q**=*s'd***Q**=*s'***P**
- Identify correct $\boldsymbol{Z_t}$ checking against next output block

Attack complexity: 2^{16} checks, with just two output blocks



Dual_EC_DRBG Observations

- No trapdoor hidden in the implementation (like for RSA and Diffie Hellman)
- Trapdoor is (can be) instead in the algorithm definition

- ***subliminally*** *(unnoticeably)*
  Trapdoor (knowledge of d) can be suspected but not proved

- ***securely (exclusively)***
  Knowledge of d needed to exploit the trapdoor

- ***forward secrecy***
  No way to run the system backwards (except solving DLP)

Dual_EC_DRBG
"Funny" facts

Randomly ordered facts about DUAL_EC_DRBG algorithm:

- It is incredibly slow (100 to 1000 times slower than the other proposals in SP800-90A)
- It is statistically biased
- It was implemented in many widely used libraries
- It was set as default CSPRNG in BSAFE by RSA (the company) and maybe other libraries
- P and Q can be chosen arbitrarily, but standard ones are mandatory for FIPS-2 validation
- Possibility of a trapdoor is widely known at least since 2007

Dual_EC_DRBG Timeline

2004	Preliminary versions
2006	Published in *NIST SP 800-90*
	NIST SP 800-90: Recommendation for Random Number Generation Using Deterministic Random Bit Generators
2007	**Shumow and Ferguson (Microsoft) show the possibility of a trapdoor**
	Other reseachers highlight potential weaknesses
2012	Maintained in *NIST SP 800-90a*
2013	**US government interception programs revealed (Snowden leakage)**
2014	**NIST recommends against its use**
2015	Withdrawn in *NIST SP 800-90a Rev. 1 in 2015*

Dual_EC_DRBG Countermeasures

- Remove n > 16 bits to produce out (e.g. 128)

  the resulting attack complexity is 2^n

  the statistical bias problem is solved

- Allow to chose different P, Q

- Or simply… do not use this generator

  Three more generators are defined in SP800-90A

  1) Hash_DRBG ┐
  2) HMAC_DRBG ┘ based on hash functions

  3) CTR_DRBG — based on block ciphers

Conclusions

- Security is not just in algorithms and protocols
- Security is not just in bug-free implementations (if any)
- Trapdoors in primitives design and in implementation can exist
- They can be very subtle and hard to detect (especially in black box model)
- Need to
  - develop in-house as many critical components as possible
  - build a trust-chain with technology providers
  - implement architectural mitigation countermeasures

Bibliography (to start with)

- Adam Young, Moti Yung, *The Dark Side of "Black-Box" Cryptography or: Should We Trust Capstone?*, Crypto 96
- Adam Young, Moti Yung, *Kleptography: Using Cryptography Against Cryptography*, Eurocrypt 97
- Adam Young, Moti Yung, *The prevalence of kleptographic attacks on discrete-log based cryptosystems*, Crypto 97
- Crepeau, Slakmon, *Simple Backdoors for RSA Key Generation*, 2003 RSA Conference
- Bernstein, Lange, Niederhagen, *Dual EC: A Standardized Back Door,* The New Codebreakers, 2004
- Shumow, Ferguson, *On the Possibility of a Back Door in the NIST SP800-90 Dual Ec Prng*, Rump Session at CRYPTO 2007
- Paterson, *Imprimitive Permutation Groups and Trapdoors in Iterated Block Ciphers,* FSE 1999

The end

Thank you!
Questions?

guglielmo.morgari@telsy.it
guglielmo.morgari@polito.it