Introduzione:

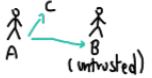
17/12

- Bitcoin gestito da noi, no da banche. Con MINING creo BITCOIN, nel 2009 1 btc = 3 cent. Perso private key, persi bitcoin!

È importante gestione secret key. • Come? La tengo per me solo? Se la perdo? Devo gestire tutto! • Usare third party? (Coinbase) Per quanto sicuri, anche loro attaccabili.

SOLUZIONE: Secret sharing, come abbiamo già visto. Usando shares (es (2,3) MULTISIG) il segreto non è mai completo.

- altra applicazione: deep Web, pagamenti in anonimo.



· A non si fida di B, e viceversa. Metto intermediario C, chi mi dice che posso fidarmi? INTRODUCO (2,3) account A,B,C, • se acquisto OK, C non interviene, faccio sign A,B per spostare denaro. Se ci sono problemi entra 'C' giudice, : con A,C sign fa rimborso, con C,B no.

C non può scappare, perché è possibile ricostruire e riavere i soldi.

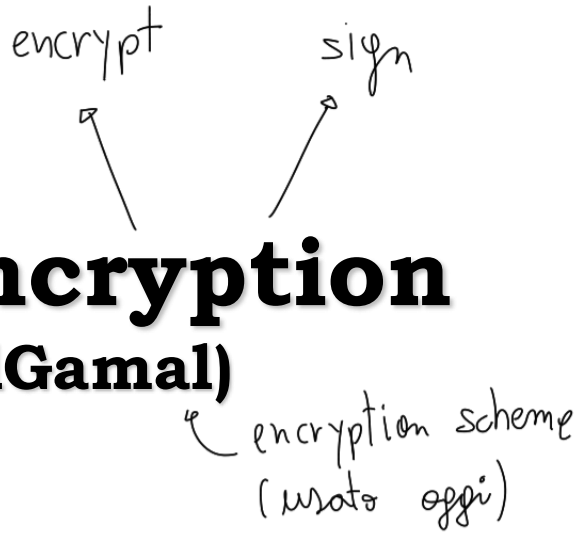
Threshold and policy-based cryptography

(Secret sharing applications and extensions)

è una crypto di "gruppo" di tipo (t,n) basata su Secret Sharing + Standard Techniques

From theory to practice

- **Practical applications of secret sharing "machinery"**
- **Several Scenarios**
  - group-oriented crypto
    - Threshold Encryption
    - Threshold Signatures
    - Fancy constructions
  - Attribute-based cryptosystems
    - (monotone) access control policies
    - More expressive than "just" thresholds
  - Delegation and distribution of authority
    - Avoid points of failure, improve resiliance
    - Capture resistance
    - Delegate security functionalities to proxies



Threshold encryption

(use-case: ElGamal)

Public Key Encryption with DLOG

➔**Recap:**

- ⇨1976: Diffie-Hellman used DLOG but «only» for key agreement
- ⇨1977: RSA solved public key encryption, but with different assumption (factorization)

➔**Question: public key encryption with DLOG → how to?** (key transfer basata su DLOG!)

➔**Answer: El Gamal, 1985**

- ⇨turn DH key agreement into an encryption protocol
- ⇨Construction: a Columbus egg! ☺

Why DLOG cryptosystems?

Conveniently implemented over Elliptic Curve Groups

And why we want EC cryptography, then?

Symm key equiv	modulus size	Elliptic Curve
80 bits	1024 bits	163 bits
128 bits	3072 bits	283 bits
256 bits	**15360** bits	571 bits

EC scales better than ordinary modular groups with increased security parameter!!

Obvious long term deployment choice!

ElGamal - background

rendo DH un cipher

➔ **Asymmetric cryptosystem**

- ⇨ Public key system, 1985, Taher Elgamal
- ⇨ Based on Discrete log
  - →Different "hard" problem than RSA (factoring)
- ⇨ Dlog based Public Key (asymmetric) cryptosystem

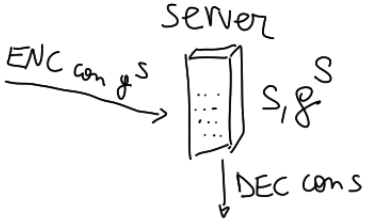
➔ **Inspired by DH**

- ⇨ DH = key agreement, NOT a cipher
- ⇨ ElGamal: turns DH idea into an asymmetric cipher

➔ **Probabilistic cipher**

- ⇨ Multiple encrypt of same message = different ciphertexts
- ⇨ Semantic security (under DDH assumption)
- ⇨ Price to pay: 1→2 ciphertext expansion

ElGamal - sketch



p large prime

g Group generator

➔ **All operations mod p**

(come DH)

Creative way to "reuse" Diffie-Hellman!

g^s, g^r : known to all (g^s public, g^r in ciphertext)

s: known only by RX (secret key)

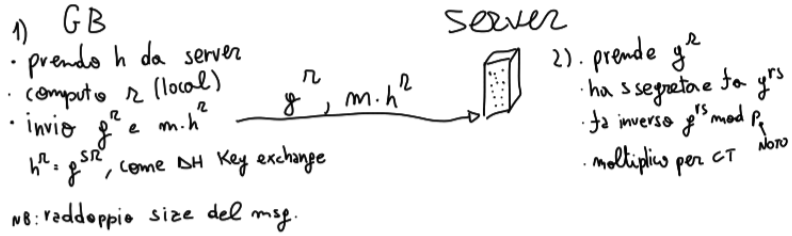
r: known only by TX (random)

g^{sr} : No way for anybody else to compute

s Private key

$h = g^s$ Public key

r *random*



mando 2 msg

Encrypt $(R, c) = (g^r, m \cdot h^r)$

Decrypt $m = c \cdot R^{-s} = \frac{c}{(g^r)^s} = \frac{m \cdot h^r}{g^{rs}} = \frac{m \cdot g^{sr}}{g^{rs}}$

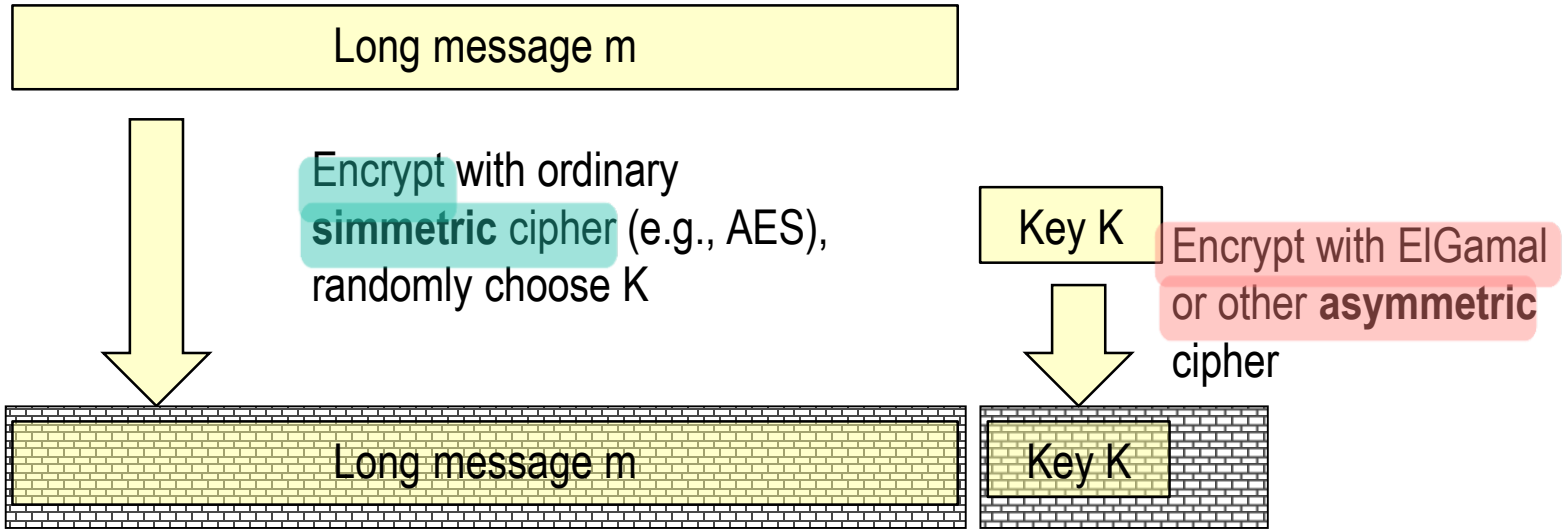
modular inverse (extended eucl. algorithm)

Anymmetric ciphers:
"hybrid" usage

➔ **Message must be point in group**

- ⇨ E.g., a number between 1 and p-1 if using mod p integers
- ⇨ Prime p: for instance 1024 bits
- ⇨ But messages can be much longer than that!

➔ **Hybrid encryption**



How El-Gamal-type crypto is used today?

ECIES = Hybrid Encryption in 5G

Elliptic Curve Integrated Encryption Scheme

➔**SUPI (IMSI) → SUCI**

(nuovo nome IMSI)

⇨Subscriber Concealed Identifier

⇨Public key encryption of SUPI (IMSI)!

⇨Use home network public key pre-loaded in SIM

➔**Approach: ECIES**

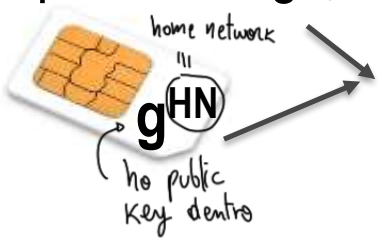
⇨Elliptic Curve Integrated Encryption Scheme

➔**Idea (without EC details):**

fanno encrypt then MAC, non usano AES-GCM

$\langle x, g^x \rangle$

Ephemeral g^x (≈ el Gamal)



$K = HKDF(g^{(HN\ x)})$ → $AES_K(SUPI)$ → HMAC

applico

$(g^{HN})^x$ el Gamal

(counter mode)

$(g^x, HMAC[AES_K(msg)])$

symmetric encryption

in entrata: prendo $g^x \rightarrow (g^x)^{HN} \rightarrow K \rightarrow$ applico ad AES

Threshold ElGamal

(msg inviati a un gruppo)

➔ **Decryption when threshold # of receivers cooperate**

- ⇨ Message can be read only by t cooperating parties (t, n)
- ⇨ Nobody has private key (!) - safer

➔ **First idea:**

- ⇨ Distribute shares of private key *s*
  - →E.g. using Pedersen for s initially unknown
- ⇨ Reconstruct s when decryption needed

(poichè serve g^s, ma stavolta parlo di gruppi, devo generare $<s, g^s>$ di Pedersen così ogni elemento ha share del segreto!)

➔ **But…**

- ⇨ Works only once!
  - →Once secret key reconstructed, ANYTHING can be decrypted!

Quando ricostruisco 's', non è più hidden! Vorrei decryption only, senza reconstruction!

➔ **Can we decrypt one message, and guarantee confidentiality for any other message?**

Towards the solution…

Encrypt $\left(g^{r_1}, m_1 \cdot h^{r_1}\right)$ $\left(g^{r_2}, m_2 \cdot h^{r_2}\right)$

r_i cambia sempre, tranne perdo S.S

Decrypt $m_1 = \dfrac{c}{\left(g^{r_1}\right)^s}$ Decrypt $m_2 = \dfrac{c_2}{\left(g^{r_2}\right)^s}$

➔ **How to decrypt m1 and NOT m2?**

➔ **How to compute denominator for r_1 s.t. computation for a different r_2 is not revealed?**

General property of exponentia tion : $A^x = A^{x_1} \cdot A^{x_2} = A^{x_1 + x_2}$

➔ **Can we interpolate "shares" at exponent?**

Getting closer…

➔**Usual polynomial** (by Pederson)

$$p(x) = s + a_1 x + a_2 x^2 + \cdots + a_{t-2} x^{t-2} + a_{t-1} x^{t-1}$$

hidden — random

➔**Usual shares**

$$(x_i, y_i) \quad y_i = p(x_i)$$

➔**Usual reconstruction formula…** (RECAP)

$$s = \sum_{shares\, x_i} y_i \Lambda_{x_i} \quad with \quad \Lambda_{x_i} = \Lambda_{x_i}(0) = \prod_{shares x_k \neq x_i} \frac{-x_k}{x_i - x_k}$$

➔**What if done at exponent?**

$$\prod A^{y_i \Lambda_{x_i}} = A^{\sum y_i \Lambda_{x_i}} = A^s$$

Actual solution

➔ **Each party owns one share** $(x_i, y_i) \quad y_i = p(x_i)$

➔ **Party gets g^r from ciphertext** $\left(g^r, m \cdot h^r\right)$

➔ **Computes as usual Lagrange coefficients**

⇨ Depend only on (known) x-axis share values $\Lambda_{x_i} = \prod_{shares x_k \neq x_i} \frac{-x_k}{x_i - x_k}$

➔ **Computes exponent share** $\left(g^r\right)^{y_i \Lambda_{x_i}}$

eseguita da elemento 'i' del gruppo $\forall i$

➔ **Sufficient # of shares permit to reconstruct decryption term**

$$\prod \left(g^r\right)^{y_i \Lambda_{x_i}} = \left(g^r\right)^{\sum y_i \Lambda_{x_i}} = \left(g^r\right)^s = g^{rs}$$

➔ **Finally factor away decryption term**

$$\frac{m \cdot h^r}{g^{rs}} = \frac{m \cdot g^{sr}}{g^{rs}} = m$$

key derivation c'è dopo, posso fare tali op. con AES

- Group El Gamal: tutti generano $s_1, s_2 \dots$, faccio Pedersen → $s = (\sum s_i) \bmod p$. Se encrypto prendo (g^r, m, g^s) preso da gruppo el gamal (pubblica)
  Se lo faccio "più semplice" ho $(g^r, m \cdot h^r)$
  r preso da tutti gli elementi: $A = R^{s_1}$, $B = R^{s_2}$, … → $\Pi = R^{\sum s_i} = R^s$ → $g^{rs} = h^r$ → $C = m \cdot \frac{h^r}{h^r} = m$
- Con Threshold scheme devo gestire Shamir, Lagrange, modulo prime (+ robusto)

Threshold signature

(use-case: RSA)

Why a threshold signature?

➔**Any t out of n members of a group can sign a message**

→Validity of (signed) message endorsed by multiple "notaries"

→Group member certified by other t members

→Place trust in more than ONE Certification Authority

→Etc

➔**If less than t members, impossible to forge a signature**

➔**Further requirements**

⇨Should reuse existing signature approaches

→(not a new signature type)

⇨Size should not "blow up" with t

RSA signature

- **Large primes p, q; N=pq**
  - ⇨ $\phi(N) = (p-1)(q-1)$
- **Pick e s.t.coprime to $\phi(N)$**
- **Compute $d = e^{-1} \bmod \phi(N)$**
  - ⇨ i.e. $e \cdot d = 1 \bmod \phi(N)$

classico approccio

- **Signing a message:**
  - ⇨ $[m, H(m)^d]$ mod N
- **Verifying signature:**
  - ⇨ $H(m) = (H(m)^d)^e$

d → faccio shares ai t players di private key

$\cdot [H(m)]^d$ expantiation

$\cdot H(M)^{\sigma_i \lambda_i} \cdot H(M)^{\sigma_{i+1} \lambda_{i+1}} \ldots$

Threshold RSA
(multi-signatures)

- Usual approach (?)
- Dealer: $f(x) = d + a_1 x + a_2 x^2 + \cdots + a_{t-2} x^{t-2} + a_{t-1} x^{t-1}$
- Share of P_i $(x_i, y_i = f(x_i)) \mod \phi(N)$
- Message to be signed $m, H(m)$
- Signature share $H(m)^{y_i \Lambda_{x_i}} \mod N$
- Valid signature $\prod H(m)^{y_i \Lambda_{x_i}} = H(m)^{\sum y_i \Lambda_{x_i}} = H(m)^d$

What's wrong????

What differs with RSA wrt previous cases?

Computing signature share…

(Devil is in the details)

➔ **Party i must compute** $H(m)^{y_i \Lambda_{x_i}} \bmod N$

↳ NON conosce $\phi(N)$, solo 'N'.

➔ **Where** $\Lambda_{x_i} = \prod_{shares x_k \neq x_i} \frac{-x_k}{x_i - x_k} = \frac{\alpha_{x_i}}{\beta_{x_i}}$ tutti interi

➔ **Hence** $H(m)^{y_i \Lambda_{x_i}} = H(m)^{y_i \alpha_{x_i} \beta_{x_i}^{-1}} \bmod N$

➔ **where** $\beta_{x_i} \cdot \beta_{x_i}^{-1} = 1 \bmod \phi(N)$ esponente, $\phi(N)$ non noto!

1) We CANNOT compute inverse without ϕ(N) → factorization needed → RSA breaks!

2) If β happens to be even (as it may well be), inverse does NOT exists

How to avoid inverses?

(brilliant remark!!)

20/12/21

esempio
- L = 7 shares = {1, 2, 4, 7}, 4 shares
- calcolo den: (4-2)(4-1)(4-7) = -18
- calcolo 4!(7-4)! = 144, divisibile per 18
- valido anche con 7 shares!
  (4-1)(4-2)(4-3)... = -36,
- se i = 4 → 4!(7-4)! = 8, ancora valido!

➔**Assume usual x_i=i, and L players**

➔**Look at Lagrange polynomial denominator**

⇨Worst case of interpolation on all L shares:

$$\Lambda_i(x) = \prod_{shares k \neq i} \frac{x-k}{i-k} = \frac{something(x)}{(i-1)(i-2)\cdots(i-(i-1))(i-(i+1))\cdots(i-L)}$$

➔**But denominator surely divides i!(L-i)!**

➔**Which in turns surely divides L! ☺**

NB: $\binom{L}{i} = \frac{L!}{i!(L-i)!}$ intero!

➔**Consequence:**

$$L! \cdot \Lambda_i(x) = \text{surely integer} = \overline{\Lambda_i}(x)$$

So what?

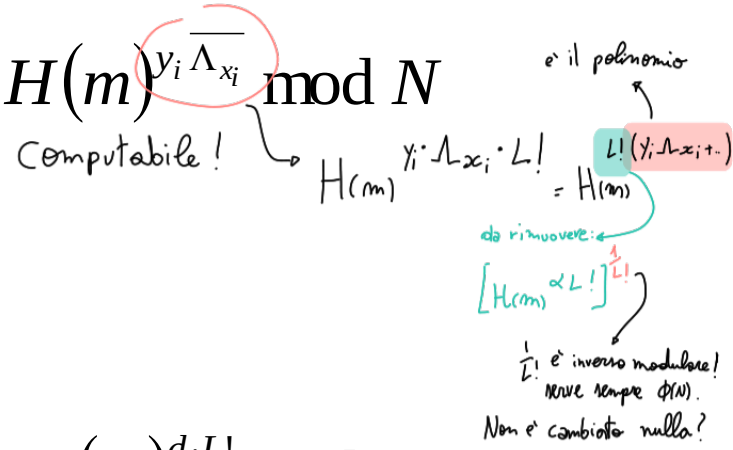
➔ **Dealer:** $f(x) = d + a_1 x + a_2 x^2 + \cdots + a_{t-2} x^{t-2} + a_{t-1} x^{t-1}$

➔ **Share i:** $(i, y_i = f(i)) \mod \phi(N)$

➔ **Compute signature share as:**

- ⇨ no inverse needed anymore
- ⇨ Largange are NOW integers

$$H(m)^{y_i \overline{\Lambda_{x_i}}} \mod N$$



➔ **Construct "signature" (?!) as**

$$\prod H(m)^{y_i \overline{\Lambda_{x_i}}} = H(m)^{L! \sum y_i \Lambda_{x_i}} = H(m)^{d \cdot L!} \mod N$$

Not yet our signature: extra factor L! at exponent…

How to get rid of this?? (cannot do mod ϕ(N) inverses: back to the start!)

Not really…

- **Remember RSA common modulus attack** (Mai riusare 'N' per Key diverse)
  - Alice encrypt message m with public key e_a
  - Bob encrypt SAME message m with different public key e_b
  - Module N is the same
  - $\gcd(e_a,e_b)=1$

Alice : $(M)^{e_1} \bmod N$

Bob : $(M)^{e_2} \bmod N$

⇨ Then

- Message can be easily decrypted using Extended euclidean algorithm:
  - » Find r,s s.t. $e_a \cdot r + e_b \cdot s = \gcd(e_a,e_b)=1$

aritmetica non modulare, sempre fattibile

interi

diverse

$\rightarrow (C_1)^{r} \cdot (C_2)^{s} = \left[(M)^{e_1}\right]^{r} \cdot \left[(M)^{e_2}\right]^{s} \bmod N$

$= M^{e_1 r + e_2 s} = M^{1} = M$

senza conoscere chiave segreta

RSA common modulus attack…

➔ **You know e_a,e_b (co-primes)**

➔ **Determine r,s from Extended Euclidean Algorithm s.t.**

$$e_a \cdot r + e_b \cdot s = 1$$

➔ **You are given encrypted messages** $m^{e_a}; m^{e_b}$

➔ **You wish to know m, but no decrypt key…**

➔ **Not needed! Decrypt message as**

$$\left(m^{e_a}\right)^r \cdot \left(m^{e_b}\right)^s = m^{e_a \cdot r + e_b \cdot s} = m$$

(cerco di usarlo a mio vantaggio!)

...used for a good purpose here!

let $L! = \Delta$

we have $H(m)^{d \cdot L!} = H(m)^{d \cdot \Delta}$ ("P_k")

msg che non conosco

$\approx$ encrypt $[H(m)^d]^{L!=e_2}$ (applico ext. end. alg.)

$C_1 = [H(m)^d]^{L!}$ (ignoto), $C_2 = H(m) = [H(m)^d]^{e_2=e}$

we want $H(m)^d = y$

but we have $H(m)^{d \cdot \Delta} = y^{\Delta}$

we also have $H(m) = H(m)^{de} = y^e$

We have cast our problem as an RSA common modulus attack!

if e and Δ are coprime, we can apply "attack"

$\Rightarrow$ and hence derive our signature $H(m)^d = y$

If public key e is a prime larger than L, then this surely works!
Besides this, RSA signature remains perfectly standard!

Riassumendo: trovare $H(m)^d \bmod N$ usando shares

- $\begin{cases} \lambda x \\ H(m)^{dL!} \end{cases}$ ricostruendo share all'esponente, sono interi, basta 'Mod N'

Ho anche:

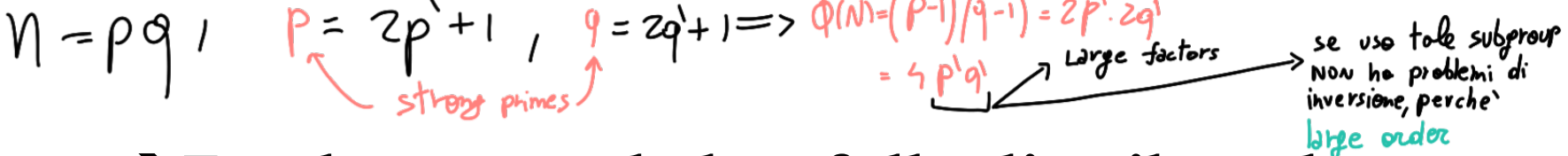
- $[H(m)^d]^e = H(m)$ (H(m) = hash) — entrambe Mod N, chiavi diverse: COMMON ATTACK

- trovo $re + sL! = 1$

- $[H(m)^d]^{r \cdot e} \cdot [H(m)^d]^{s \cdot L!} = [H(m)^d]^{re + sL! \equiv 1} \bmod N$

Caveats

- **Previous presentation largely simplified**
  - Actual proposal by Victor Shoup, 2000, Practical threshold signatures
    - Runs computations on special subgroup (quadratic residues) and relies on strong RSA (safe primes)
    - explicitly includes verifiability and security proofs
  - However, essence captured by our simplified explanation



- **Further extended as fully distributed**
  - Foque+Stern, 2001 è possibile distribuire, usando 'd' nascosta in φ(N) ... molto avanzato!
  - Complexity overcome: hard to distributely generate strong RSA module
    - Bypassed in Foque/Stern work by alternative approach