Crittografia asimmetrica: ElGamal e Rabin

Luca Cittadini

Alessio Campisano

Claudio Sasso

Introduzione allo schema di cifratura El Gamal

Luca Cittadini

Classi di complessità e riduzioni

- Classe P:
  - Insieme dei problemi risolubili da un calcolatore in tempo polinomiale rispetto alle dimensioni dell'input
- Classe NP:
  - Insieme dei problemi risolubili da un calcolatore non-deterministico in tempo polinomiale rispetto alle dimensioni dell'input
  - Sappiamo che qualsiasi problema in NP è risolubile in tempo al più esponenziale
  - Tuttavia, né P=NP, né P!=NP sono state dimostrate
- Riduzione polinomiale
  - Algoritmo per trasformare, in tempo polinomiale, una istanza di un problema A in una istanza di un problema B
  - Permette di risolvere A se siamo in grado di risolvere B
  - Ci interessano solo riduzioni polinomiali (Karp-riduzioni)

Alcuni problemi notevoli nell'ambito della crittografia

Problema	**Descrizione**
FATTORIZZAZIONE	*Problema della fattorizzazione intera*: dato un intero positivo n, trovare la sua fattorizzazione $n = p_1^{e1} p_2^{e2} \quad p_k^{ek}$ (dove p_i sono numeri primi distinti e gli e_i interi positivi ≥ 1)
SQROOT	*Radice quadrata modulo n*: da un intero composto n e $a \in Q_n$ (l insieme dei redsiui quadratici modulo n), trovare una radice quadra di a modulo n; cioè un intero x tale che $x^2 \equiv a (\bmod n)$.
DLP	*Problema del logaritmo discreto*: dato un numero primo p, un generatore $\alpha \in Z^*_p$ e un elemento $\beta \in Z^*_p$, trovare l intero x, $0 \leq x \leq p-2$, tale che $\alpha^x \equiv \beta (\bmod p)$.
GDLP	*Problema del logaritmo discreto generalizzato*: dato un gruppo ciclico finito G di ordine n, un generatore $\alpha \in G$ e un elemento $\beta \in G$, trovare un intero x, $0 \leq x \leq n-1$, tale che $\alpha^x = \beta$.
DHP	*Problema Diffie-Hellman*: dato un numero primo p, un generatore $\alpha \in Z^*_p$ e gli elementi $\alpha^a \bmod p$ e $\alpha^b \bmod p$, trovare $\alpha^{ab} \bmod p$.

Osservazione sulla complessità

- Cosa si intende per “dimensione dell’input”?
- Per i numeri interi, due prospettive diverse:
  - Dimensione = grandezza del numero
  - Dimensione = numero di cifre necessarie per rappresentare il numero, in qualsiasi base
- Nota: il numero di cifre necessarie per rappresentare un numero n è $\log(n)$.
- Nella complessità computazionale, la dimensione dell’input è quasi sempre considerata come la seconda alternativa proposta
- Anche in crittografia, si parla usualmente della **lunghezza** della chiave (in bit), piuttosto che della sua grandezza

El Gamal: Generazione delle chiavi

- Visione ad alto livello:
  - Ogni utente *u* genera un numero primo a caso, p, trova un generatore g di Z_p^*, ed un numero a casuale, compreso tra *1* e *p-2*
  - La chiave pubblica dell'utente consiste nella tripla $(p, g, g^a \bmod p)$
  - La sua chiave privata consiste nel numero a

El Gamal: schema di cifratura

- Per inviare un messaggio cifrato all'utente *u*, dobbiamo:
  - Ottenere la chiave pubblica di *u,* ovvero la tripla ($p, g, g^a \bmod p$)
  - Codificare il messaggio come un intero m in Z_p.
  - Scegliere un intero k a caso, $1 \leq k \leq p-2$
  - Calcolare $x = g^k \bmod p$
  - Calcolare $y = m\,(g^a)^k \bmod p$
- Il messaggio cifrato *c* è dato dalla coppia

$$\boxed{(x,y)}$$

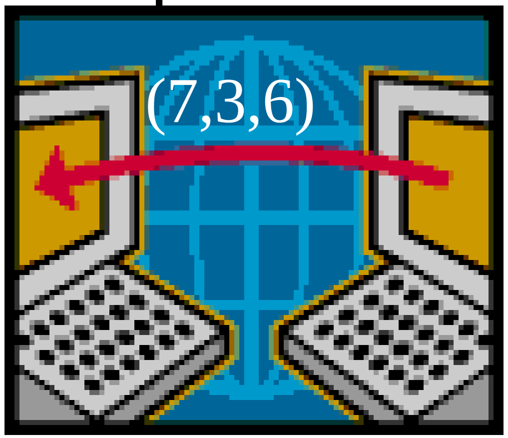
El Gamal: schema di decifrazione

- L'utente *u* per decifrare il messaggio criptato (x,y), deve:
  - Calcolare, usando la sua chiave privata a, il numero x^{-a}.
  - Recuperare m, calcolando $m=x^{-a}\, y \mod p$
- Infatti, $x^{-a}\, y = \cancel{(g^k)^{-a}}\;\; m\; \cancel{(g^a)^k} =$

$$m \bmod p$$

Capire El Gamal: un esempio

- Generiamo una nostra coppia di chiavi.
  - Prendiamo p=7, g=3 generatore, a=3
  - Calcoliamo g^a mod p : $3^3 = 6 \bmod 7$
  - Abbiamo le chiavi!
  - Teniamo segreta 3, ma pubblichiamo (7,3,6)

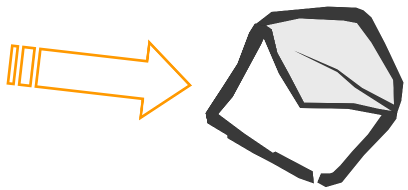


Capire El Gamal: un esempio

- Un nostro amico vuole dirci il suo numero civico: 4
  - Sceglie a caso un numero tra 1 e 6; ad esempio, 2
  - Calcola 3^2 = **2** mod 7
  - Calcola $4{\cdot}6^2$ = **4** mod 7
  - Ci invia il messaggio

2

4



Capire El Gamal: un esempio

- Riceviamo il messaggio (2,4)
- Usiamo la chiave per calcolare 2^{-3}:
  - $2^{-1} = 4 \bmod 7$ (infatti $8 = 7+1$)
  - $2^{-3} = 4^3 = 1 \bmod 7$
- Moltiplichiamo questo numero per 4…
  - $1 \cdot 4 = 4 \bmod 7$
- Purtroppo abbiamo scelto k tale che $g^{ak}=1$
  - È un'eventualità che può sempre avvenire, con probabilità $1/p$
- Quindi la seconda componente del messaggio cifrato è in realtà proprio il messaggio in chiaro!

Requisiti dello schema El Gamal: logaritmo discreto

- Affinchè tutto funzioni, è necessario che:
  - Sia computazionalmente impossibile ottenere dalla chiave pubblica la chiave privata. Questo è (apparentemente) garantito dalla difficoltà di calcolare $a = \log_g(g^a) \bmod p$
  - Sia relativamente semplice, e soprattutto veloce, calcolare gli esponenziali discreti

    g^k e $(g^a)^k$

Problema del logaritmo discreto e cifratura di El Gamal

- Il problema del logaritmo discreto è un problema che è <u>ampiamente riconosciuto</u> come difficile
  - Si conoscono algoritmi generici esponenziali, ed algoritmi specifici polinomiali, che funzionano solo se i dati del problema soddisfano determinati criteri aggiuntivi
  - <u>Non è stato dimostrato</u> nessun lower bound esponenziale per il problema generico
  - Per di più, non è stato dimostrato il fatto che El Gamal sia legato esclusivamente al logaritmo discreto.
    - Potrebbe cioè esistere un sistema per ottenere *m* data la chiave pubblica ed il testo cifrato, <u>senza</u> dover ricorrere al calcolo del logaritmo discreto

El Gamal: prestazioni

- Affinchè lo schema sia praticamente usabile occorre che:
  - sia facile, e ragionevolmente veloce, calcolare (x,y) per chi invia il messaggio. Notare come per ogni messaggio inviato devono essere calcolate due esponenziali discrete
    - g^k e $(g^a)^k$
    - che però sono indipendenti dal messaggio
    - possono essere calcolate in anticipo (o “offline”) , se necessario

El Gamal: sicurezza

- Un ruolo importante è giocato dal numero *k*, scelto a caso dal mittente
  - Questo garantisce che cifrando più volte uno stesso messaggio in chiaro, otterremo “ogni volta” testi cifrati diversi
  - Non proprio ogni volta, la probabilità di collisione aumenta ad ogni cifratura dello stesso messaggio

El Gamal: sicurezza

- D'altra parte, la presenza dell'intero k comporta la necessità di avere un testo cifrato lungo il doppio rispetto al messaggio in chiaro
  - infatti bisogna trasmettere due interi di dimensioni paragonabili g^k e $m(g^a)^k$
  - in genere, avranno bisogno di un numero di cifre simile (almeno in termini di ordini di grandezza)
- Questo fenomeno è chiamato <u>message expansion</u>

Randomized encryption

- Lo schema El Gamal è uno dei tanti che fanno uso della casualità nel processo di cifratura.
- L'idea fondamentale è aumentare la sicurezza crittografica
  - Azzerando o diminuendo l'efficacia di attacchi di tipo chosen plaintext, attraverso un mapping uno-a-molti tra testo in chiaro e testo cifrato
  - Azzerando o diminuendo l'efficacia di attacchi di tipo statistico
    - Modifica la distribuzione di probabilità dell'input

Osservazioni e discussioni

- È possibile che tutti gli utenti del sistema utilizzino parametri comuni
  - Possono condividere lo stesso numero primo p e lo stesso generatore g
    - In questo caso la chiave pubblica è più corta
    - È necessaria qualche precauzione in più, che discuteremo più avanti

Osservazioni e discussioni

- In pratica, possiamo considerare il sistema El Gamal come una variante dello schema Diffie-Hellman
  - Una chiave, k, è generata casualmente e trasmessa nel messaggio cifrato (g^k)
  - Entrambe le parti ottengono una chiave segreta di sessione (g^{ak}). Il mittente possiede g^a e lo eleva a k; il destinatario possiede g^k e lo eleva ad a.
  - Meccanismo praticamente identico allo scambio delle chiavi di Diffie-Hellman

Osservazioni e discussioni

- Come scegliere un buon numero primo *p*?
  - Compromesso tra sicurezza e prestazioni
  - Infatti, al crescere di *p* è sempre più difficile calcolare il logaritmo discreto di un certo numero
    - Ma non è detto che El Gamal possa essere forzato solo attraverso il logaritmo discreto
  - Inoltre, al crescere di *p* possiamo mandare un numero maggiore di messaggi
    - Infatti il messaggio è un intero *m* in Z_p
    - La pratica più comune consiste nel dividere il messaggio in blocchi e poi cifrare ogni blocco
      - In tal caso, al crescere di *p* cresce la dimensione del blocco

Osservazioni e discussioni

- Ma un numero primo molto grande può essere problematico
  - Difficile (ovvero costoso) trovare numeri primi grandi (numeri primi "titanici")
  - Difficile trovare un generatore del gruppo Z_{p}^{*} se p è molto grande
  - Difficile calcolare gli esponenziali discreti

Osservazioni e discussioni

- Se k fosse fissato?
  - Se k non fosse casuale, presi due messaggi m_1 ed m_2 e i relativi testi cifrati (x,y_1) e (x,y_2), avremmo $$\frac{y_1}{y_2} = \frac{m_1 \cdot \cancel{g^{ak}}}{m_2 \cdot \cancel{g^{ak}}} = \frac{m_1}{m_2}$$
  - Saremmo in grado di risalire a m_2 conoscendo m_1!!
    - Una volta noto un blocco, potremmo decifrarli tutti quanti
    - Vulnerabilità ad attacchi known ciphertext

Osservazioni e discussioni

- Se non usassimo un generatore?
  - Allora il compito del criptoanalista sarebbe facilitato
  - Spazio di ricerca dei logaritmi più piccolo: non tutti i numeri hanno il loro logaritmo discreto se la base non è un generatore
  - Alcuni valori non comparirebbero MAI come risultato di g^k mod p!
    - Spazio dei messaggi cifrati più piccolo
  - Ma soprattutto… esisterebbero più chiavi private valide!! Infatti senza generatore ogni numero ha più di un logaritmo
    - Aumenta la probabilità di trovare almeno una soluzione

Osservazioni e discussioni

- Esempio dell'uso di un "non generatore" *g'* in Z_7^*:
  - Usiamo 2. La chiave pubblica potrebbe essere (p=7, g'=2,$(g')^a$=4), con chiave privata a=2 oppure a=5
  - Infatti $2^2 = 4 \bmod 7$
  - Ma anche $2^5 = 4 \bmod 7$
  - Aumentano le probabilità di trovare una soluzione!!

Osservazioni e discussioni

- Se volessimo usare un gruppo diverso da Z_p^*?
  - Lo schema El Gamal può essere generalizzato ad un qualsiasi gruppo G
    - Non necessariamente strutture algebriche basate sull'aritmetica modulare
  - Alcuni gruppi sono realmente usati in pratica:
    - Gruppi basati su polinomi irriducibili
    - Gruppi basati sui punti delle curve ellittiche

Precauzioni particolari: algoritmo index-calculus

- Il miglior algoritmo conosciuto per il calcolo dei logaritmi discreti
  - Cerca innanzitutto una “base di fattori”, ovvero un sottoinsieme S del gruppo G tale che buona parte degli elementi in G possono essere scritti come prodotto di elementi in S
  - Calcola i logaritmi degli elementi di S attraverso sistemi di equazioni lineari
  - Ricostruisce infine il logaritmo desiderato combinando opportunamente quelli degli elementi in S

Precauzioni particolari: algoritmo index-calculus

- Algoritmo probabilistico!
- Una volta trovato S, comincia a calcolare $g^r \bmod p$, dove r è un numero casuale
  - Se è vero che la maggior parte dei numeri in G possono essere fattorizzati con elementi in S, <u>probabilmente</u> $$\boxed{g^r = s_1^{e_1} \cdot \ldots \cdot s_n^{e_n}, \quad s_1, \ldots, s_n \in S}$$
  - Posso quindi scrivere $r = e_1 \log(s_1) + \ldots + e_n \log(s_n)$
  - È un'equazione <u>lineare</u>!
  - Trovandone abbastanza possiamo costruire un sistema determinato e calcolare i logaritmi degli elementi in S

Precauzioni particolari: algoritmo index-calculus

- Una volta noti i logaritmi degli elementi in S
  - Dato un qualsiasi numero b, se possiamo fattorizzarlo con elementi in S

$$\boxed{b = s_1^{e_1} \cdot \ldots \cdot s_n^{e_n}, \quad s_1, \ldots, s_n \in S}$$

  - Allora, facendo logaritmo ad entrambi i membri,

$$\boxed{\log b = e_1 \log s_1 + \ldots + e_n \log s_n}$$

- Calcoliamo qualsiasi logaritmo come combinazione lineare dei logaritmi degli elementi in S

Precauzioni particolari: algoritmo index-calculus

- La maggior parte del lavoro computazionale dell'algoritmo è dedicato a generare S
  - E risolvere i logaritmi degli elementi in S
- Tali elementi dipendono dal gruppo ciclico in questione
  - e non dal numero di cui si cerca il logaritmo!
  - $a = \log_g(g^a) \bmod p$ dipendono da p, non da a!
    - Se p è condiviso, c'è il rischio di un attacco "in parallelo" a TUTTE le chiavi del sistema
    - Una volta generato un buon sottoinsieme S, e calcolati i logaritmi dei suoi elementi, un gran numero di chiavi sono rotte in tempo breve

Parametri usati al giorno d’oggi

- Abbiamo bisogno di numeri primi molto grandi
  - 512 bit sono già da tempo considerati a rischio
  - 768 bit erano la raccomandazione minima nel 1996
  - Oggi almeno 1024 bit
  - Ancora più bit se i parametri sono condivisi da tutti gli utenti.

El Gamal: Schema della firma digitale

Alessio Campisano

Sommario

- Obbiettivi della firma elettronica
- Schema della firma digitale
- Tassonomia
  - Appendix
  - Recovery
  - Randomized
  - Deterministic
- El Gamal: Schema di firma digitale
  - Generazione chiave
  - Algoritmo: Processi di firma e verifica
  - Discussione sulla sicurezza e sulle prestazioni

Firma digitale(1)

- Lo scopo della firma elettronica è garantire un legame tra informazioni (in formato digitale) e le rispettive entità che le hanno generate o divulgate
- Al fine di garantire le seguenti proprietà:
  - Autenticazione
  - Autorizzazione
  - Integrità dei dati
  - Non ripudiabilità

Firma digitale(2)

- La firma digitale di un messaggio non è altro che una sequenza di bit (un numero) che dipende da un’informazione conosciuta soltanto dal firmatario (chiave segreta) e dal contenuto del messaggio stesso
- Il processo di firma consiste nel trasformare il messaggio e le informazioni segrete possedute dal firmatario in un tag

Un po' di nomenclatura…

- *M* è l'insieme dei messaggi da firmare
- *S* è l'insieme degli elementi chiamati firme, costituiti da stringhe binarie di lunghezza fissa
- S_A rappresenta una una funzione da *M* a *S* eseguita dall'entità *A* per firmare i propri messaggi. Questo processo detto **"signing transformation"** è tenuto segreto da *A* per garantire che altre entità non possano prendere il suo posto
- V_A è una trasformazione che va dal *MxS* all'insieme {true,false} ed è detta **"verification transformation"**. Questa funzione è pubblica ed è utilizzata dalle altre entità per verificare le firme generate da A

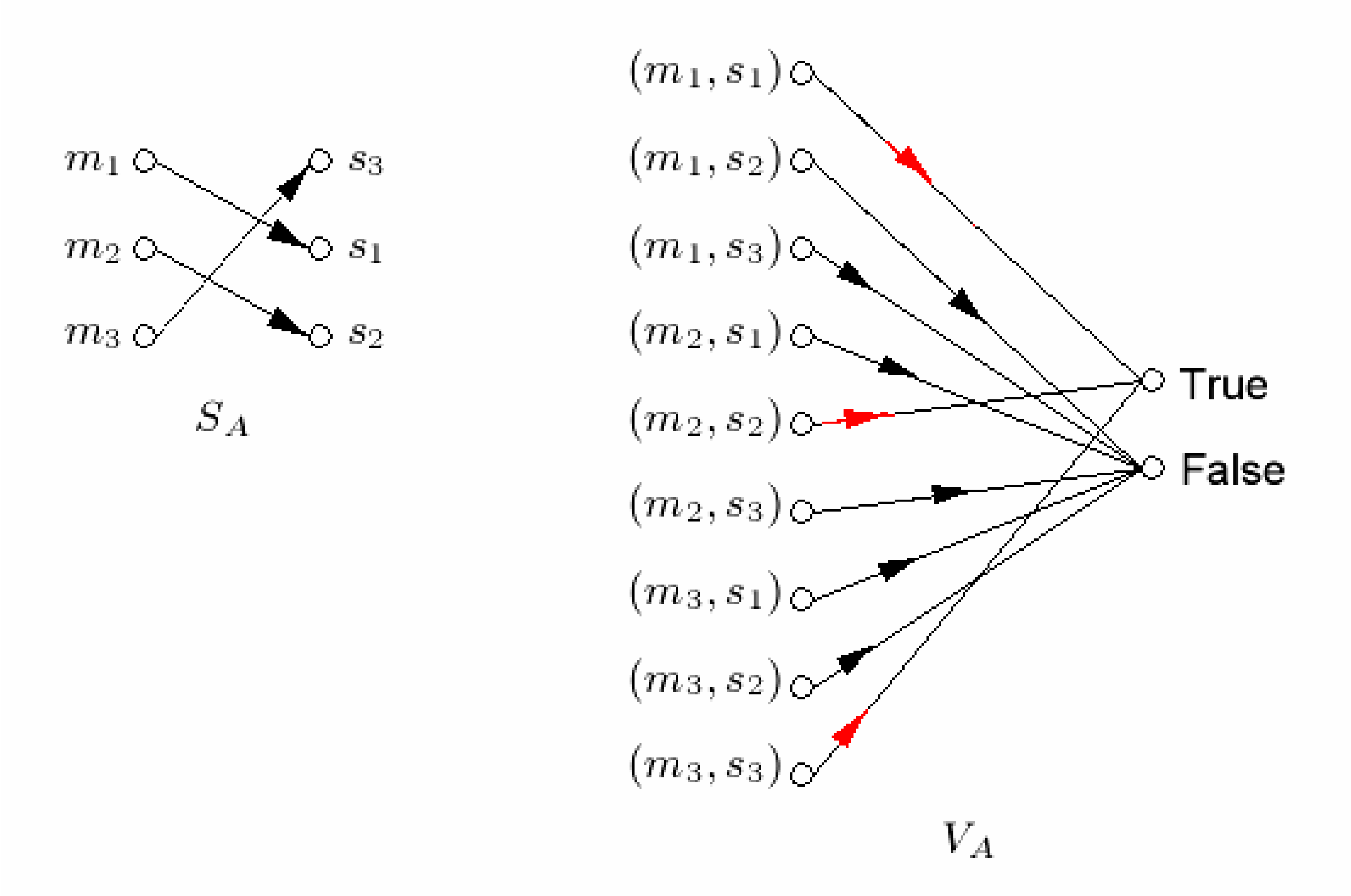
Schema firma digitale

- Le funzioni:
  - **S_A “signing transformation”**
  - **V_A “verification transformation”**

  costituiscono la base dello schema di firma digitale per l’entità A

Schema di firma digitale: un semplice esempio

- M = {m1,m2,m3} S = {s1,s2,s3}



Come funziona uno schema di firma digitale?

- Signing procedure
  - L'entità A (il firmatario) crea una firma per il messaggio $m \in M$ svolgendo le seguenti operazioni
    1. Calcola $s=S_A(m)$
    2. Trasmette la coppia (m,s)
- Verification procedure
  - Per verificare che una firma s di un messaggio m sia stata effettivamente creata dall'entità A, l'entità B (il destinatario) deve:
    1. Ottenere la funzione di verifica V_A (pubblica) di A
    2. Computare $u=V_A(m,s)$
    3. Accettare il messaggio soltanto se u=true

Proprietà delle funzioni di firma e verifica

1. *s* è una firma valida per A sul messaggio *m* se e solo se $V_A(m,s)=true$
   - come è possibile vedere dall'esempio mostrato in precedenza
2. Non è possibile per un'entità diversa da A trovare un messaggio $m \in M$ e una firma $s \in S$ tale che $V_A(m,s)=true$
   - garantisce che l'identità di A possa essere associata soltanto ad A stesso senza possibilità di equivoci
   - questa proprietà non è stata formalmente dimostrata anche se ci sono diversi schemi di firma digitale, che si ritiene soddisfino questa proprietà

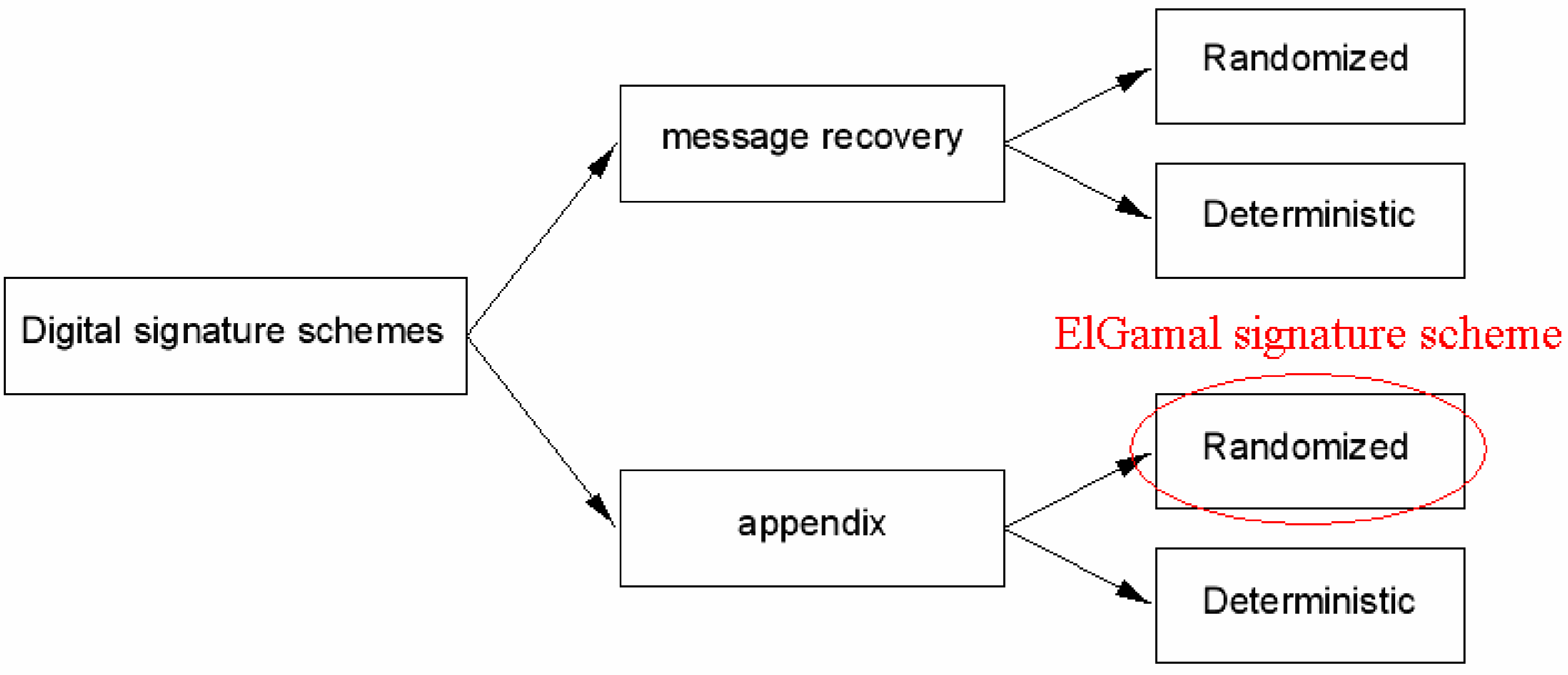
Classificazione degli schemi di firma digitale(1)

- Digital signature schemes with appendix
  - Hanno bisogno del messaggio originale come input per verificare la validità della firma
- Digital signature schemes with message recovery
  - Non hanno bisogno del messaggio originale come input della funzione di verifica. Il messaggio è recuperato dalla dalla firma stessa

Classificazione degli schemi di firma digitale(2)

- Digital signature schemes deterministic
  - Presentano sempre la stessa esecuzione ogni volta che vengono richiamati con lo stesso input
- Digital signature schemes randomized
  - Seguono delle decisioni casuali ad ogni esecuzione, a fronte dello stesso input possono generare sequenze di esecuzione ogni volta differenti

Tassonomia degli schemi di firma digitale



El Gamal:
Schema di firma digitale

- Lo schema di firma di El Gamal ricade nella classe degli schemi **Appendix-Randomized** e può essere applicato a messaggi in formato binario di lunghezza variabile

- Richiede l'utilizzo di una funzione hash così definita:

$$h : \{0,1\}^* \rightarrow Z_p$$

dove p è un numero primo abbastanza grande

El Gamal: Generazione delle chiavi

- Ciascuna entità A crea una chiave pubblica e la corrispondente chiave privata:
  1. A genera un numero primo a caso, p, trova un generatore α di Z_p^*
  2. Sceglie un numero a casuale,tale che tra $1 \leq a \leq p-2$
  3. *Calcola* $y = \alpha^a \bmod p$
  4. La chiave pubblica dell'utente consiste nella tripla (p,α,y)
  5. La sua chiave privata è formata invece dal numero a

El Gamal: Generazione della firma

- L'entità A che intende firmare il proprio messaggio deve:
  1. Scegliere un numero casuale k; $1 \leq k \leq p-2$ e primo con $p-1$ ($mcd(k,p-1)=1$)
  2. calcolare $r = \alpha^k \bmod p$
  3. computare $k^{-1} \bmod (p-1)$

     (l'inverso di k è sempre calcolabile)
  1. risolvere l'espressione $s = k^{-1} \{h(m)-ar\} \bmod (p-1)$
  2. La firma di A per il messaggio m è *costituita dalla coppia* (r,s)

El Gamal:
Verifica della firma

- Per verificare la firma di A su m, B dovrebbe:
  1. Ottenere la chiave pubblica di A, la terna (p, α, y)
  2. Controllare che r rispetti il vincolo $1 \leq r \leq p-1$, ed in caso contrario rifiutare il messaggio
  3. Calcolare $v_1 = y^r r^s \bmod p$
  4. Eseguire la funzione hash sul messaggio e ottenere $h(m)$ per calcolare $v_2 = \alpha^{h(m)} \bmod p$
  5. La firma è accettata soltanto se $v_1 = v_2$

$V_1=V_2$?

- E' sufficiente ammettere che se $v_1=v_2$ allora il documento è stato firmato da A?

  Sì

- Vediamo perché…
  - Ricordiamo che $v_1 = y^r r^s \bmod p$ e

    $v_2 = \alpha^{h(m)} \bmod p$ dunque se $v1 = v2$ allora

    $y^r r^s \bmod p = \alpha^{h(m)} \bmod p$
  - Inoltre $s \equiv k^{-1} \{h(m)-ar\} \bmod (p-1)$ e moltiplicando entrambi i membri per k otteniamo

    $sk \equiv \{h(m)-ar\} \bmod (p-1)$ da cui

    $h(m) \equiv \{ar + sk\} \bmod (p-1)$
  - Questo implica che

    $\alpha^{h(m)} \equiv \alpha^{ar+ks} \equiv (\alpha^a)^r (\alpha^k)^s \pmod p \equiv y^r r^s \pmod p$

Un pò di matematica…

- Notare che l'ultimo punto della dimostrazione precedente

  $$\alpha^{h(m)} \equiv \alpha^{ar+ks} \equiv y^r r^s \ (mod\ p)$$

  deriva da

  $$h(m) \equiv \{ar + sk\} \ mod\ (p\text{-}1)$$

- Questo è garantito dalla seguente definizione:

  Se $r \equiv s \ (mod\ p - 1)$ allora per ogni a intera

  $a^r \equiv a^s \ (mod\ p)$

  (Handbook of Applied Cryptography by A. Menezes, P.van Oorschot and S. Vanstone.- Fact 2.127(ii))

El Gamal: Esempio firma digitale(1)

- Generazione delle chiavi
  - A sceglie un numero primo $p=2357$ ed un suo generatore $\alpha=2$ di Z^*_{2357} ed un numero casuale a compreso tra 0 e $p-2$ che costituirà la sua chiave privata $a=1751$
  - Infine A calcola

    $y = \alpha^a \bmod p = 2^{1751} \bmod 2357 = 1185$
  - Come ultimo passo A rende nota la propria chiave pubblica, la terna $(p=2357, \alpha=2, y=1185)$

El Gamal: Esempio firma digitale(2)

- Generazione della firma
  - Supponiamo per facilitare le cose che il messaggio sia in Z_p e che la funzione hash non venga utilizzata, dunque, $h(m) = m$ (nella realtà non è possibile effettuare questa semplificazione)
  - Per firmare il messaggio $m=1463$ A deve scegliere un numero $k=1529$ (gcd(k,p-1)=1) e calcolare

    $r = \alpha^k \bmod p = 2^{1529} \bmod 2357 = 1490$
  - Calcolare l'inverso di k

    $k^{-1} \bmod (p-1) = 245$
  - L'ultimo passo che resta da fare è il calcolo di s

    $s = k^{-1} \{h(m)-ar\} \bmod (p-1) =$

    $= 245\{1463-1751(1490)\} \bmod 2356 = 1777$

El Gamal:
Esempio firma digitale(3)

- Verifica della firma
  - A questo punto l'entità B per verificare la firma ha bisogno della chiave pubblica di A *($p=2357$,$\alpha=2$,$y=1185$)* del messaggio *$m=1463$* (ricordiamo che lo schema è di tipo: Appendix) e della firma del messaggio costituita dalla coppia *($r=1490$,$s=1777$)*
  - B calcola la funzione hash del messaggio (che in questo caso è data dall'identità del messaggio stesso per le ipotesi fatte in partenza) *$h(m) = 1463$ e*

    $v_1 = y^r r^s \bmod p = 1185^{1490} 1490^{1777} \bmod 2357 = \boxed{1072}$
  - B calcola ahche

    $v_2 = \alpha^{h(m)} \bmod p = 2^{1463} \bmod 2357 = \boxed{1072}$
  - Alla fine B accetta il messaggio di A

Algoritmo per il calcolo dell'inverso in Z_n

- INPUT: $a \in Z_n$.
- OUTPUT: a^{-1} mod n, ammesso che esista

1. Si utilizza l'algoritmo di Euclide modificato per trovare x e y tali che

   $d = ax+ny$

   con $d = mcd(a,n)$

2. Se $d>1$, allora l'inverso non esiste, altrimenti a^{-1} è proprio uguale a x

El Gamal: Sicurezza dello schema di firma(1)

- Un estraneo che intendesse forgiare la firma di A su un messaggio *m* potrebbe selezionare un numero casuale *k* e calcolare $r = \alpha^k \bmod p$ (e fin qui tutto bene)
- Il problema risiede infatti nel calcolo di

  $s = k^{-1} \{h(m)-ar\} \bmod (p-1)$
- Data la difficoltà del calcolo del logaritmo discreto al malintenzionato non rimane che scegliere come valore *s* un numero casuale, con una probabilità di successo pari a *1/p* che per *p* abbastanza grande tende a zero

El Gamal: Sicurezza dello schema di firma(2)

- Per garantire l'efficacia della firma e fare in modo che entità estranee non riescano a venire a conoscenza della propria chiave privata, A dovrebbe firmare i propri messaggi scegliendo un valore di k sempre differente altrimenti…
  - $s_1 = k^{-1}\{h(m_1)-ar_x\} \bmod (p-1)$
  - $s_2 = k^{-1}\{h(m_2)-ar_x\} \bmod (p-1)$
  - $(s_1 - s_2)k \equiv \{h(m_1)-h(m_2)\} \bmod (p-1)$ *e nel caso in cui* $(s_1 - s_2) \not\in [0] \bmod (p-1)$
  - $k = (s_1 - s_2)^{-1}\{h(m_1)-h(m_2)\} \bmod (p-1)$
- Determinato k sarebbe banale risalire alla chiave privata *a* dell'entità A da s_1 oppure da s_2

El Gamal: Sicurezza dello schema di firma(3)

- L'utilizzo di una funzione hash è indispensabile per evitare che estranei riescano ad assumere l'identità di A ingannando altre entità.
- In particolare se non viene utilizzata una funzione hash lo schema è soggetto ad un attacco definito existential forgery attack che può essere portato a termine nel modo seguente
  - Un entità esterna sceglie una coppia di interi *(u,v)* con $gcd(v,p-1)=1$
  - Calcola

    $r = \alpha^u y^v \bmod p = \alpha^{u+av} \bmod p \quad (y = \alpha^a \bmod p)$

    $s = -rv^{-1} \bmod (p-1)$

continua…

El Gamal: Sicurezza dello schema di firma(3)

- In questo modo la coppia *(r,s)* rappresenta una firma per il messaggio

  $m = su \bmod p$
- Infatti se effettuiamo una verifica del messaggio (come nell'esempio precedente) confrontando

  $v_1 = y^r r^s \bmod p$ e $v_2 = \alpha^m \bmod p$

  $(y=\alpha^a \bmod p,\ r = \alpha^{u+av} \bmod p,\ s = -rv^{-1} \bmod (p-1))$
- Otteniamo:

  $(\alpha^a)^r (\alpha^{u+av})^{-rv^{-1}} = \alpha^{u(-rv^{-1})} = \alpha^{su} = \alpha^m$
- Essendo $v_1 = v_2$ il messaggio *m=su* è visto dagli altri utenti come se fosse stato effettivamente firmato dal legittimo proprietario della firma e non da un estraneo.

El Gamal: Sicurezza dello schema di firma(4)

- Un altro elemento molto importante dell'algoritmo di verifica della firma è rappresentato dal secondo punto
  2. Controllare che r rispetti il vincolo $1 \leq r \leq p-1$, ed in caso contrario rifiutare il messaggio
- Nel caso l'utente non eseguisse questo controllo lo schema sarebbe soggetto ad un ulteriore tipo di attacco.
- Un avversario che fosse a conoscenza di un messaggio m e della relativa firma (r,s) generata dall'entità A, sarebbe in grado di rompere la firma di A

El Gamal: Sicurezza dello schema di firma(4)

- Scelto un messaggio *m'* e calcolato *h(m')* l'estraneo potrebbe calcolare

  $u = h(m')[h(m)]^{-1} mod(p-1)$

  (assumendo che $[h(m)]^{-1}$ *esista)*

  $s' = su \bmod (p-1)$ e *r'* tale che siano verificate le due congruenze seguenti $r' \equiv ru \bmod (p-1)$ e

  $r' \equiv r \bmod p \; (\Rightarrow r' > p)$

  (l'esistenza di *r'* è garantita dal Teorema del Resto Cinese)
- La coppia *(s',r')* è una firma per il messaggio *m'*

Teorema del Resto Cinese

- Se gli interi $n_1, n_2, \ldots, n_k$ sono primi tra loro, allora il sistema di congruenze

$$x \equiv a_1 \pmod{n_1}$$
$$x \equiv a_2 \pmod{n_2}$$
$$.$$
$$.$$
$$.$$
$$x \equiv a_k \pmod{n_k}$$

ammette sempre una soluzione.

El Gamal: Prestazioni dello schema di firma (1)

- Il processo di generazione della firma è molto soddisfacente in termini di prestazioni infatti richiede soltanto il calcolo:
  1. di un’esponenziale discreto
  2. dell’inverso di un numero
  3. di due moltiplicazioni modulari
- In particolare le prime due operazioni possono essere eseguite in momenti diversi (off-line) rispetto alla terza quindi la complessità dell’algoritmo può essere ridotta alle due sole moltiplicazioni

El Gamal: Prestazioni dello schema di firma (2)

- La verifica della firma è computazionalmente più onerosa, richiede infatti il calcolo di tre esponenziali discreti
  - Ciascun esponenziale richiede $3/2\lceil lg\ p \rceil$ moltiplicazioni discrete in media, per un totale di $9/2\lceil lg\ p \rceil$ moltiplicazioni
  - Modificando l'algoritmo di verifica, calcolando $v_1 = y^r r^s \alpha^{-h(m)} \bmod p$ e accettando la firma solo nel caso in cui $v_1 = 1$
  - Il costo dell'algoritmo è ridotto a $15/8\lceil lg\ p \rceil$ moltiplicazioni, grazie alla possibilità di calcolare gli esponenziali simultaneamente

Complessità delle operazioni elementari in Z_n

Operation		Bit complexity
Modular addition	$(a+b) \bmod n$	$O(\lg n)$
Modular subtraction	$(a-b) \bmod n$	$O(\lg n)$
Modular multiplication	$(a \cdot b) \bmod n$	$O((\lg n)^2)$
Modular inversion	$a^{-1} \bmod n$	$O((\lg n)^2)$
Modular exponentiation	$a^k \bmod n, k < n$	$O((\lg n)^3)$

El Gamal:
Varianti dello schema di firma

- Lo schema di El Gamal presenta diverse varianti che differiscono dall'originale principalmente per il calcolo della funzione

  $s = k^{-1} \{h(m)-ar\} \bmod (p-1)$

- In particolare se la riscriviamo in questo modo

  $u = av + kw \bmod (p-1)$

  possiamo riassumere le varianti dell'algoritmo in un'unica tabella

El Gamal: Varianti dello schema

	u	v	w	Signing equation	Verification
1	$h(m)$	r	s	$h(m) = ar + ks$	$\alpha^{h(m)} = (\alpha^a)^r r^s$
2	$h(m)$	s	r	$h(m) = as + kr$	$\alpha^{h(m)} = (\alpha^a)^s r^r$
3	s	r	$h(m)$	$s = ar + kh(m)$	$\alpha^s = (\alpha^a)^r r^{h(m)}$
4	s	$h(m)$	r	$s = ah(m) + kr$	$\alpha^s = (\alpha^a)^{h(m)} r^r$
5	r	s	$h(m)$	$r = as + kh(m)$	$\alpha^r = (\alpha^a)^s r^{h(m)}$
6	r	$h(m)$	s	$r = ah(m) + ks$	$\alpha^r = (\alpha^a)^{h(m)} r^s$

Schema di codifica di Rabin

Claudio Sasso

Introduzione

- Variante di RSA proposta da Rabin nel 1979
- Si è *quasi certi* che RSA sia equivalente al problema della fattorizzazione
  - Ma si potrebbe forzare RSA trovando un metodo alternativo alla fattorizzazione
- Si è **certi** che del legame fra problema di fattorizzazione e crittosistema di Rabin
  - Soltanto riuscendo a risolvere il problema della fattorizzazione in tempo polinomiale deterministico si può pensare di rompere il crittosistema di Rabin
  - Non vi possono essere metodi alternativi
- Ciò aumenta la sicurezza del sistema

Basi Matematiche

- Il crittosistema di Rabin si basa sul problema del calcolo della radice modulo n di un numero.
  - Calcolare il quadrato modulo n è un’operazione semplice
  - Calcolare la radice modulo n è un’operazione difficile a meno che non si abbia un indizio sui fattori primi che compongono il numero in questione
- Come per tutti i problemi legati a crittosistemi, è un one way trapdoor:
  - P in un senso
  - NP nell’ altro

Radici modulo n

- Sia Z^*_n un gruppo moltiplicativo generico.
- $a \in Z^*_n$ ammette una radice modulo ***n*** se esiste un altro numero $x \in Z^*_n$ tale che

$$x^2 \equiv a \bmod n$$

- L'insieme di tutti $a \in Z^*_n$ che ammettono radice quadrata modulo n viene indicato con Q_n
- L'insieme di tutti $a \in Z^*_n$ che **non** ammettono radice quadrata modulo n viene indicato con $\overline{Q_n}$

Radici modulo n

- La definizione è analoga a quella delle radici in N (ovvero analoga alla definizione delle radici di numeri interi)
- La complessità del problema deriva ovviamente dalla struttura dei campi finiti (che sono ciclici)

- Dalla definizione precedente, risulta che

$$0 \notin Q_n \quad 0 \notin \overline{Q_n}$$

Cardinalità degli insiemi Q

- Dal teorema precedente si può dedurre la cardinalità di Q_n e di $\overline{Q_n}$

$$|Q_n| = \frac{n-1}{2} \qquad |\overline{Q_n}| = \frac{n-1}{2}$$

- Ovvero, **in un gruppo ciclico soltanto la metà degli elementi ammette radici modulo n**
- L'altra metà non ha radici modulo n

Radici Quadrate modulo *p*

- Sia $a \in Q_p$
- Se $x \in Z^*{}_p$ è tale che $x^2 \equiv a \bmod p$ allora *x* è una **radice quadrata** di a modulo *p*

- Se *p* è primo e $a \in Q_p$ allora *a* ha esattamente 2 radici quadrate in $Z^*{}_p$

Radici Quadrate modulo n

- Consideriamo il caso di *n* qualunque
  - Non più esclusivamente primo
- Possiamo fattorizzare *n*: $n = p_1^{e_1} \cdot p_2^{e_2} \cdot .... \cdot p_k^{e_k}$
- Dove $p_1, p_2, ..., p_k$ sono *k* primi distinti
- Dove $e_1, e_2, ..., e_k$ sono *k* esponenti tutti maggiori di 1

- Se $a \in Q_n$ allora ha *a* esattamente 2^k radici quadrate in $Z^*{}_n$, tutte distinte

Radici modulo ***p*** nel caso particolare $p \equiv 3 \bmod 4$

- La metà degli elementi di Z_p^* ammette radici modulo *p,* e se p è primo tutti gli $a \in Q_p$ hanno ***esattamente 2 radici modulo p***
- Nel caso particolare in cui $p \equiv 3 \bmod 4$
  - Se ***a*** ammette due radici modulo p, esse saranno *x e –x*, e ***-a*** *non ammetterà alcuna radice modulo p*
  - Se ***-a*** ammette due radici modulo p, esse saranno sempre *x e –x*, e ***a*** *non ammetterà alcuna radice modulo p*

Radici modulo *p* nel caso particolare $p \equiv 3 \bmod 4$

- Il nostro obiettivo è quello di estrarre le radici di un $a \in Q_p$, ovvero di risolvere l'equazione:
$$x^2 = a \bmod p$$
- Se $p \equiv 3 \bmod 4$ (e quindi può essere riscritto come $p \equiv 4k - 1$),allora le soluzioni di questa equazione sono:

$$x_1 = a^k \bmod p \qquad x_2 = -x_1$$

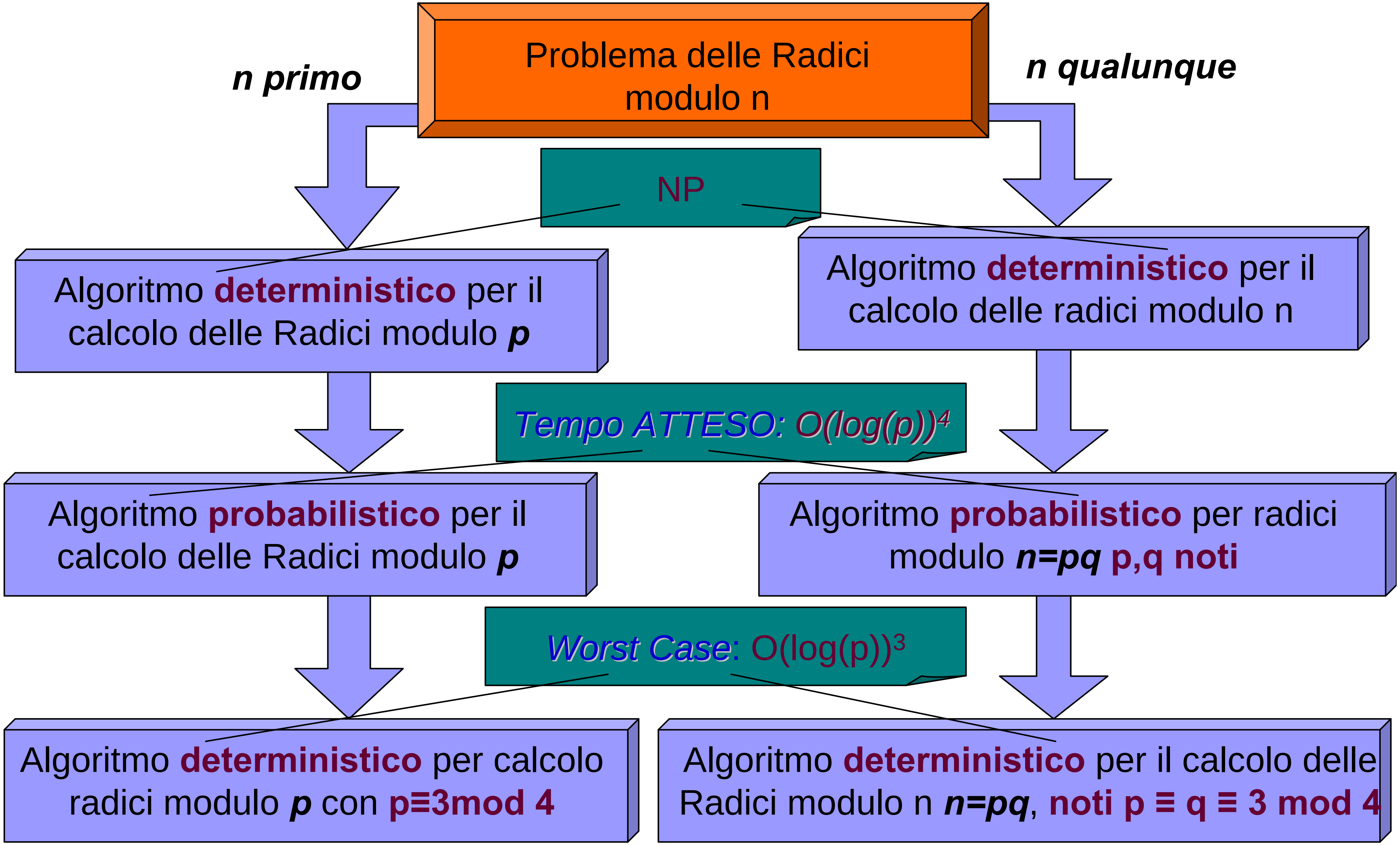
Il problema delle radici in Z_n^*

- Il nostro obiettivo è quello di estrarre le radici di un $a \in Q_n$, ovvero di risolvere l'equazione:
$$x^2 = a \bmod n$$
- Nel caso in cui n non è primo, il calcolo è computazionalmente molto costoso
- Nel caso in cui n è primo, il calcolo è meno costoso ?
  - Sappiamo che la metà degli elementi di Z_n^* hanno esattamente 2 radici, l'altra metà non ne ammette nessuna
  - Ma nonostante questo…

Il problema delle radici in Z_p^*

- **Non esiste <u>nessun</u> algoritmo deterministico in grado di trovare le due radici quadrate modulo *p* di un intero *a* in <u>tempo polinomiale, anche se *p* è primo</u>**
- Si utilizzano algoritmo in cui si effettua una scelta casuale
- Per questi algoritmi è possibile calcolare *un tempo d'esecuzione atteso*
- Ora illustreremo un algoritmo per il calcolo delle radici nel caso in cui p è primo

Algoritmi per il calcolo delle radici



Algoritmo calcolo radici di ***a*** modulo ***p*** con ***p* primo** (1)

- Input: ***a*** un numero dispari compreso fra ***1*** e ***p-1***
- Output: le due radici di ***a*** in $Z^*{}_p$

Se ***a*** non ammette radici in $Z^*{}_p$

<u>return -1</u>

else

ripeti

b = random(1**,*****p-1)***

finchè non trovi un ***b che ammette radici in*** $Z^*{}_p$

Tramite divisione per 2, scrivi ***p-1*** come $r \cdot 2^k$ con ***r*** dispari

Algoritmo calcolo radici di ***a*** modulp ***p*** con ***p* primo** (2)

Sia $c = b^{r} \bmod p$

Sia $t = a^{(r+1)/2} \bmod p$

per i che va da 1 a k-1

$$d = \left(t^{2} \cdot a^{-1}\right)^{2^{k-i-1}} \bmod p$$

if $d = -1 \bmod p$ *allora* $t = t \cdot c \bmod p$

$$c = c^{2} \bmod p$$

<u>return t, -t</u>

Considerazioni sull'algoritmo

- Il calcolo di b è casuale
  - Viene estratto un numero casuale compreso fra 1 e p-1, fintanto che non soddisfi una proprietà
    - Nel caso peggiore, potremmo dover scandire tutto l'intervallo!
- E' comunque un algoritmo Las Vegas, dato che non sbaglia mai
  - Alcuni algoritmi con scelte casuali possono sbagliare
- Il suo ***tempo atteso*** di esecuzione è

$$O(\log p)^4$$

Specializzazione per $p \equiv 3 \bmod 4$

- In un passo dell'algoritmo dobbiamo riscrivere p-1 come il prodotto fra un numero dispari e una potenza di 2.
- Nel caso in cui ***p ≡ 3 mod 4*** allora ***p -1 ≡ 2 mod 4*** e quindi ***r = 1***
- ***L'algoritmo si riduce ad un unico passo:***
  - ***Sia*** $t = a^{(p+1)/4} \bmod p$
  - ***return (t,-t)***
- Complessità stabilita da quell'unica operazione:

$$O(\log p)^3$$

Algoritmo calcolo radici di ***a*** modulo ***n*** **non primo**

- Ora dobbiamo esaminare il calcolo delle radici modulo ***n*** di un intero ***a***
  - In questo caso ***n non è primo***
- Tale calcolo è risolvibile efficientemente soltanto se si conoscono i fattori di n
- Nel nostro caso immaginiamo che tali fattori siano ***p*** e ***q***
- Le cose migliorano ancora se $p \equiv q \equiv 3 \bmod 4$
  - Complessità lineare

Algoritmo calcolo radici di ***a*** modulo ***n non primo***

- Input: ***a*** un numero dispari compreso fra ***1*** e ***n-1,*** ed **i fattori di *a,* ovvero *p* e *q***
- Output: le due radici di ***a*** in Z^*_n

- Siano ***r***, ***-r*** le radici di ***a*** modulo ***p***
- Siano ***s***, ***-s*** le radici di ***a*** modulo ***q***
- Calcola (con l'algoritmo di Euclide esteso) due interi, ***d*** e ***c*** tali che $dq + cp = 1$
- Sia $x = (rdq + scp) \bmod n$; $y = (rdq - scp) \bmod n$
- ***Return*** $(\pm x \bmod n, \pm y \bmod n)$

Considerazioni

- Conoscendo i fattori di ***n***, basta calcolare i le radici di ***a*** modulo ***p*** e ***a*** modulo ***q***, e combinarle opportunamente
- Tale calcolo predomina in complessità rispetto all'algoritmo di euclide esteso
- Quindi la complessità di questo algoritmo dipende dal calcolo delle radici modulo ***p*** e ***q***
- E nel caso in cui $p \equiv q \equiv 3 \bmod 4$?

Radici di ***a*** modulo ***n*** *con* ***n =pq***

$$p \equiv q \equiv 3 \bmod 4$$

- In questo caso l'algoritmo si semplifica
- Stessi Input/Output
  - Questa volta $p \equiv q \equiv 3 \bmod 4$

- Sia $r = a^{(p+1)/4} \bmod p$
- Sia $s = a^{(q+1)/4} \bmod p$
- Calcola (con l'algoritmo di Euclide esteso) due interi, ***d*** e ***c*** tali che $dq + cp = 1$
- Sia $x = (cps + dqr) \bmod n$
- Sia $y = (cps - dqr) \bmod n$
- ***Return*** $(\pm x \bmod n, \pm y \bmod n)$

Complessità

- Se ***n*** non è primo, e non si conoscono i suoi fattori primi, questo problema è in NP
- Se ***n*** non è primo e si conoscono i fattori, la sua complessità dipende da quella dell'algoritmo per il calcolo delle radici modulo ***p*** con ***p*** primo
- Tale algoritmo, realizzato probabilisticamente, ha un valore attesto del tempo di esecuzione logaritmico rispetto a ***p***
- Se però siamo nel caso $p \equiv 3 \bmod 4$ allora l'algoritmo per il calcolo delle radici modulo ***p***, con ***p*** primo, ha complessità logaritmica
- Quindi, se $p \equiv 3 \bmod 4$ e $q \equiv 3 \bmod 4$, l'algoritmo per il calcolo delle radici di ***n***, con ***n = pq***, ha complessità polinomiale deterministica
- **Quindi A saprà decriptare il mesaggio di B in tempo polinomiale deterministico**

Algoritmo a chiave pubblica di Rabin

- La matematica alla base di questo algoritmo è sostanzialmente quella presentata fino ad ora
  - Anche se molti risultati interessanti sono stati omessi, ed altri sono stati semplificati
- L'algoritmo a chiave pubblica di Rabin utilizza i due algoritmi visti in precedenza
  - Anche se si deve tener conto del problema della non unicità delle radici

Algoritmo a chiave pubblica di Rabin

- Due soggetti A, B vogliono comunicare
- Entrambi generano (o hanno a disposizione) una chiave pubblica una chiave privata
  - La chiave pubblica è un generico intero n
  - La chiave privata è la coppia (p,q) tale che

    $n = pq$
    - p, q sono due primi

Fase di Codifica

- B vuole criptare il messaggio m che invierà ad A
  - B ottiene la chiave pubbica di A, n
  - Considera il messaggio come un intero compreso fra 0 ed $n-1$
    - Ovvero, come un intero di Z_n^*
  - Calcola $c = m^2 \bmod n$
  - Invia c ,ovvero il testo cifrato, ad A
- Questo schema deve essere raffinato
- Sembra un caso particolare di RSA, con $e = 2$
  - In realtà 2 non è coprimo con $\varphi(n)$ quindi questo esponente non sarebbe valido per RSA

Fase di Decodifica

- A questo punto A deve risalire al messaggio m a partire dal testo cifrato c
  - A ha proprio il problema di calcolare le radici di c modulo n
  - Tali radici saranno 4!
    - Come detto precedentemente, un intero modulo ***n***, scomponibile in ***k*** fattori primi, ha esattamente 2^k radici
    - In questo caso, ***n*** è scomponibile in 2 fattori, ovvero ***p*** e ***q***
  - A potrà calcolare le radici di ***c*** modulo ***n*** in ***<u>tempo atteso</u>*** polinomiale deterministico dato che conosce ***p*** e ***q***.
  - Basterà usare l'algoritmo illustrato precedentemente

Fase di Decodifica

- Rimangono 2 problemi

1) Come fa A a capire quale delle 4 radici è quella giusta?
2) Nel caso peggiore, in cui la scelta casuale fatta nell'algoritmo per il calcolo delle radici di p e q, non aiuta, A potrebbe trovarsi di fronte ad un problema troppo complesso da risolvere

Raffinamenti

- E’ possibile risolvere entrambi i problemi.

1) **B** introduce una ridondanza nel messaggio grazie alla quale A può capire univocamente quale delle 4 alternative è valida

- Generalmente si replicano gli ultimi 64 bit del messaggio

2) ***p*** e ***q*** generati da A devono essere tali che

$$p \equiv q \equiv 3 \bmod 4$$

- In questo modo A saprà sicuramente calcolare in tempo polinomiale deterministico le radici di ***c***.

Sicurezza

- Immaginiamo che un malintenzionato M voglia intercettare la comunicazione di B, e scoprire cosa ha trasmesso ad A
- M deve fare soltanto una cosa per conoscere la comunicazione: estrarre le radici di c modulo n
- Sappiamo che è un problema difficile, e che, per come è stato affrontato da noi, dipende dalla fattorizzazione di n, la chiave pubblica di B
- Ma chi ci garantisce che M non conosca un modo diverso per calcolare le radici di c modulo n ?

Sicurezza

- Tale garanzia ci è data da queste due affermazioni:

$$\textbf{\textit{SQROOT}} <_P \textbf{\textit{FACTORING}}$$

$$\textbf{\textit{FACTORING}} <_P \textbf{\textit{SQROOT}}$$

- Esiste una Karp-riduzione polinomiale dal problema della fattorizzazione di ***n*** a quello del calcolo delle radici modulo ***n***, e ***viceversa***.
- Malintenzionato M dovrebbe saper risolvere in tempo polinomiale il problema della riduzione per rompere il cifrario
  - Ma a quel punto Malintenzionato M avrebbe messo in crisi tutto il mondo della crittografia

Sicurezza

- La considerazione precedente ci mette al riparo da malintenzionati passivi
  - Ovvero che non possono interagire con i soggetti comunicanti
- E per un malintenzionato attivo?
  - Può scegliere cosa mandare ad A
- Se non si usa la ridondanza, un attacco di tipo chosen-chipertext è fatale all'algoritmo di Rabin
- Se si usa la ridondanza, tale tipo di attacco ha molte meno probabilità di riuscita

Ultime considerazioni

- Rabin ha esposto questo schema come “teorico”
  - Interessante perché è il primo per cui si dimostra la stretta correlazione col problema della fattorizzazione
- Ma lo schema di Rabin a cui venga aggiunta la ridondanza è di grande interesse pratico
  - Resiste bene agli attacchi
  - E’ efficiente: è estremamente veloce in quanto coinvolge soltanto elevamenti al quadrato, abbastanza facili da effettuare
    - In fase di codifica è più veloce di RSA
    - In fase di decodifica la sua velocità è comparabile a RSA

Firma digitale

- E’ possibile pensare ad uno schema di firma che faccia uso delle considerazioni esposte sin ora sul problema delle radici
- Generazione delle chiavi:
  - Un’autorità può generare le chiavi in questo modo:
    - Sceglie due primi p, q
    - Calcola $n = pq$
  - n è la chiave pubbica, p q la chiave privata

Firma digitale

- Con l’algoritmo di Rabin, lo spazio delle firme è Q_n
- Definiamo una funzione che va dallo spazio dei messaggi M a quello delle firme:

$$R : \mathrm{M} \rightarrow Q_n$$

  - Tale funzione è biiettiva e conosciuta da tutti

- Un’entità A può firmare un messaggio ***m*** con una chiave ***n*** effettuando queste operazioni:
  - Calcola $\overline{m} = R(m)$
  - Calcola una radice di $\overline{m}$: $s = SQROOT(\overline{m} \bmod n)$
  - ***s*** è la firma del messaggio

Firma digitale

- Per verificare la firma ***s*** e recuperare il messaggio ***m***, B deve:
  - Ottenere ***n***, chiave pubblica di A
  - Calcolare $\underline{m} = s^2 \bmod n$
  - Verificare che $\underline{m} \in M_R$
    - Se non è così, la firma non è valida
  - Recuperare ***m***: $m = R^{-1}(\underline{m})$