

# Crittografia a chiave pubblica: uno sguardo alle vulnerabilità di RSA e Diffie-Hellman



Leonardo Alfредucci

Relatori

Dott. Gaspare Ferraro

Prof.ssa Anna Bernasconi

Università di Pisa

Dipartimento di Informatica

Pisa, 7 ottobre 2022

# Indice

- 1 Introduzione
- 2 RSA
- 3 Diffie-Hellman su campo primo
- 4 Diffie-Hellman su curve ellittiche
- 5 Conclusioni



# Parte 1

## Introduzione



# Introduzione

- Una grandissima quantità di informazioni viaggia attraverso la rete: è dunque di fondamentale importanza proteggere i dati che vengono scambiati.
- Si passeranno in rassegna i due protocolli più usati per lo scambio di chiave: RSA e Diffie-Hellman, quest'ultimo analizzato su campo primo e su curve ellittiche.
- Lo scopo della tesi è quello di andare al di là di una trattazione teorica di questi due protocolli, concentrandosi piuttosto sull'aspetto pratico.



# Parte 2

## RSA



# RSA: la teoria dietro al protocollo

- È un cifrario asimmetrico. Sono dunque presenti due coppie di chiavi:
  - $(e, n)$  utilizzata per cifrare (*chiave pubblica*);
  - $(d, n)$  utilizzata per decifrare (*chiave privata*).
- Si scelgono due numeri primi  $p$  e  $q$ .
- Si calcola  $n = p \cdot q$  e  $\phi(n) = (p - 1) \cdot (q - 1)$ .
- Si sceglie  $e < \phi(n)$  tale che  $\gcd(e, n) = 1$ .
- Si calcola  $d = e^{-1} \bmod \phi(n)$ .
- Tutti i passi descritti possono essere svolti in tempo polinomiale.



# RSA: cifratura e decifrazione

- Per cifrare un messaggio  $m$  è sufficiente calcolare il crittogramma  $c$  come:

$$c = m^e \mod n.$$

- Per ottenere il messaggio  $m$  dato  $c$  è sufficiente calcolarlo come:

$$m = c^d \mod n.$$



# RSA: uno sguardo alla sicurezza

- La sicurezza di RSA è garantita grazie al problema della fattorizzazione di un numero  $n$  come prodotto di due fattori  $p \cdot q$ .
- Per questo è importante scegliere due fattori primi molto grandi, tale che il modulo sia almeno 2048 bit, meglio ancora se 3072 bit.
- Nel 1999 è stato fattorizzato RSA-512 in circa 7 mesi utilizzando centinaia di calcolatori e impiegando l'equivalente di 8400 anni di CPU.
  - Nel 2009 lo stesso attacco poteva essere effettuato in 83 giorni da un solo calcolatore.
- Nel 2020 il numero più grande fattorizzato ha 829 bit, impiegando l'equivalente di 2700 anni di CPU.





# RSA: un'analisi sperimentale sulla sicurezza

- Sono stati implementati tre algoritmi per la fattorizzazione:
  - *Wheel factorization*: fondamentalmente un brute force sul numero, cercando i divisori;
  - *Pollard's rho factorization*: di natura probabilistica, è quello più efficiente;
  - *Fermat factorization*: è più veloce se i due numeri primi sono vicini tra loro.
- Sono stati fattorizzati moduli da 120 bit utilizzando l'algoritmo *Pollard's rho* in poco meno di un'ora su un moderno calcolatore.



# RSA: l'esponente pubblico $e$

- L'esponente pubblico non dovrebbe essere troppo grande per velocizzare la cifratura.
- Con l'*Algoritmo delle quadrature successive*, l'operazione può essere svolta in tempo  $O(\log_2 e + hm(e))$ , dove  $hm(e)$  rappresenta il *peso di Hamming*.
  - Il peso di Hamming rappresenta il numero di simboli diversi dal simbolo 0 dell'alfabeto utilizzato.
- L'esponente pubblico, dato che non contiene alcuna informazione, viene generalmente riutilizzato per molteplici operazioni.



# RSA: valori più utilizzati di $e$ con i rispettivi *pesi di Hamming*

<i>X.509</i>			<i>PGP</i>			<i>Combinati</i>		
$e$	$hm(e)$	%	$e$	$hm(e)$	%	$e$	$hm(e)$	%
65537	2	98.4921	65537	2	48.8501	65537	2	95.4933
17	2	0.7633	17	2	39.5027	17	2	3.1035
3	2	0.3772	41	3	7.5727	41	3	0.4574
35	3	0.1410	19	3	2.4774	3	2	0.3578
5	2	0.1176	257	2	0.3872	19	3	0.1506
7	3	0.0631	23	4	0.2212	35	3	0.1339
11	3	0.0220	11	3	0.1755	5	2	0.1111
47	5	0.0101	3	2	0.0565	7	3	0.0596
13	3	0.0042	21	3	0.0512	11	3	0.0313
65535	16	0.0011	$2^{127} + 3$	3	0.0248	257	2	0.0241
altri	-	0.0083	altri	-	0.6807	altri	-	0.0774



# RSA: gli schemi di padding

- Prima di essere cifrato mediante RSA, ogni messaggio viene modificato con gli schemi di padding.
- Gli schemi di padding sono importanti in crittografia:
  - aggiungono una componente di casualità;
  - non rendono possibile un recupero anche parziale del messaggio, fissandone univocamente la lunghezza.
- Due degli schemi più utilizzati sono *PKCS1 v1.5* ed *OAEP*.



# RSA: malleabilità senza il padding

- Il padding aiuta ad evitare che RSA sia *malleabile*.
  - Ad esempio, se un attaccante conosce  $c = m^e \bmod n$ , che non utilizza il padding, può sostituire  $c' = c \cdot 2^e \bmod n$ .
  - Quando  $c'$  verrà decifrato, si otterrà  $2m$  invece che l'originario  $m$ .
- Con il padding questa modifica molto semplice non è più possibile.



# RSA: generazione errata della chiave

- L'esponente  $e$  deve essere scelto coprimo con  $\phi(n)$ .
- In una pre-release di Windows 10, non veniva effettuato il controllo che  $\gcd(e, \phi(n)) = 1$  nel momento in cui veniva scelto l'esponente pubblico.
- Il corretto funzionamento di RSA è compromesso e la decifrazione non è più possibile.



# RSA: la probabilità di scegliere l'esponente pubblico errato

- Ma quanto spesso questo problema si verifica nella pratica?
- Per  $e = 65537$ , la probabilità  $P$  che  $e|(p-1)$  oppure  $e|(q-1)$  è data da

$$P < \frac{1}{32000}$$

- Il caso  $e^i \mid (p-1) \cdot (q-1)$  con  $i > 2$  non verrà trattato.
- Windows 10 è utilizzato da oltre un miliardo di dispositivi.
  - Più di 30000 utenti coinvolti.



# RSA: il recupero dei messaggi erroneamente cifrati

- Se la chiave viene generata in modo errato ad ogni crittogramma potrebbero corrispondere e messaggi che lo generino.
- E se i messaggi perduti sono importanti?
- Il recupero dei messaggi è esponenziale nella dimensione di  $e$ .
  - Per fortuna,  $e$  viene generalmente scelto basso.
  - Per  $e = 65537$  il recupero dei messaggi con un moderno calcolatore avviene in circa 30 secondi.





# RSA: scartare i messaggi durante il recupero

- Come si possono scartare i messaggi automaticamente?
- Analizzando il contesto.
- Analizzando i messaggi che rispettano le caratteristiche degli schemi di padding:
  - con *OAEP*, ci si aspetta solo un risultato (il messaggio originario);
  - con *PKCS1 v1.5* ci si aspetta almeno un falso positivo.



# RSA: esponente pubblico corto

- Un bug imponeva  $e = 1$ .
- Si deve prestare attenzione che intervenga la riduzione in modulo.
- In generale moduli corti possono facilitare gli attacchi basati sui reticoli.



# RSA: moduli ripetuti

- È comune che uno stesso modulo  $n$  sia condiviso tra più host.
  - Il 4% dei moduli usati in HTTPS risulta condiviso tra più host.
  - Il 60% delle chiavi SSH e il 65% di quelle usate per IPv4 risultano condivise.
  - Non è una vulnerabilità se gli host non sono correlati.
- Molti router e dispositivi della stessa linea di un produttore condividono lo stesso modulo: si possono decifrare i testi a vicenda.
- A causa di problemi con il PRNG e con i seed iniziali molte chiavi sono risultate uguali.



# RSA: un fattore in comune

- Se un primo è condiviso tra due moduli  $n_1$  e  $n_2$  è possibile trovare facilmente l'altro primo come  $\gcd(n_1, n_2)$ .
- Esistono dataset pubblici contenenti moduli RSA, per verificare se uno dei due primi è condiviso.



# RSA: vulnerabilità DROWN

- Nella sua prima versione sfruttava il semplice schema di padding *PKCS1 v1.5*.
- Si può effettuare se il server manda un errore se il padding è errato.
- Per questo attacco sono necessarie milioni di query.
  - Sfruttando il messaggio di errore un attaccante può restringere i possibili valori assunti da  $m$ .
- Si stima che nel 2016 il 33% dei siti HTTPS fossero vulnerabili a questo attacco.



# RSA: conclusioni

- Al giorno d'oggi esistono alternative migliori per lo scambio della chiave.
- RSA non è sicuro per utilizzi post-quantistici.
  - L'*Algoritmo di Shor* sui computer quantistici permette di calcolare la fattorizzazione in tempo polinomiale probabilistico.
- Ridurre l'utilizzo di RSA è difficile per ragioni di retrocompatibilità.
  - Di TLS v1.3 è stata ritardata l'uscita perché al suo interno ha eliminato lo scambio di chiavi basato su RSA.



## Parte 3

# Diffie-Hellman su campo primo



# DH su campo primo: la teoria dietro al protocollo

- È un protocollo per lo scambio di chiave.
- $A$  e  $B$ , che vogliono comunicare, si devono accordare su due parametri:
  - un numero primo  $p$ ;
  - un generatore  $g$  di  $\mathbb{Z}_p^*$ .
- Per lo scambio di chiave:
  - $A$  e  $B$  scelgono due interi casuali  $a, b$  tali che  $1 < a, b < p - 1$ ;
  - $A$  calcola  $n_A = g^a \bmod p$ ,  $B$  calcola  $n_B = g^b \bmod p$ ;
  - $A$  e  $B$  si scambiano rispettivamente  $n_A$  ed  $n_B$ ;
  - $A$  calcola  $k = n_B^a \bmod p$ ,  $B$  calcola  $k = n_A^b \bmod p$ ;
  - $k$  è condivisa poiché  $k = g^{a \cdot b} \bmod p$ .





# DH su campo primo: uno sguardo alla sicurezza

- Un attaccante per decifrare la comunicazione dovrebbe calcolare il logaritmo discreto:

$$a = \log_g n_A \quad \text{oppure} \quad b = \log_g n_B.$$

- Ad oggi non sono noti algoritmi eseguire questa operazione in tempo polinomiale.
- L'attacco migliore è l'*Index calculus*, che ha complessità  $O(2^{\sqrt{b \cdot \ln b}})$ .
- $p$  dal 2013 dovrebbe essere almeno di 2048 bit.
  - Nel 2015 quasi tutti i server HTTPS, IPsec e SSH supportavano primi a 1024 bit.
  - Il calcolo dellogaritmo discreto con  $p$  a 530 bit è stato effettuato nel 2007.
- Questo protocollo è soggetto ad attacchi *man-in-the-middle*.



# DH su campo primo: accordo sui parametri del gruppo

- Negli ultimi standard i parametri del gruppo vengono scelti da una lista.
- Se l'ordine  $q$  di  $g$  ha sottogruppi  $q_i$  si può effettuare il calcolo in modulo per i sottogruppi e infine usare il *Teorema cinese del resto*.
  - I primi ritenuti sicuri hanno la forma  $p = 2q + 1$ , con  $q$  primo, e si usa un generatore  $g$  del sottogruppo di ordine  $q$ .



# DH su campo primo: moduli non primi

- In alcune implementazioni  $p$  non è un primo.
- È una vulnerabilità solo se  $p$  ha fattori primi molto piccoli.
  - Si potrebbe usare il *Teorema cinese del resto*.
  - Il calcolo del logaritmo discreto è difficile tanto quanto il problema della fattorizzazione.



# DH su campo primo: valori da evitare

- Sottogruppi di ordine 1 e 2 sono da evitare come scambi di chiave.
- Se  $g$  genera l'intero gruppo, allora sono presenti i sottogruppi di ordine 1 e 2.
  - Se viene scambiato il valore 1, il segreto condiviso non può che essere 1.
  - Se i primi sono generati come spiegato precedentemente per essere *sicuri* ( $p = 2q + 1$ ) allora non contengono questi valori.



# DH su campo primo: un bit insicuro

- Con il sottogruppo di ordine 2 la situazione è più grave.
  - $B$  è un host che ricicla la stessa chiave Diffie-Hellman per molteplici connessioni.
  - $A$  (l'attaccante) manda a  $B$  il valore di scambio  $n_A = -1$ .
  - La chiave condivisa non può che essere 1 o  $-1$ .
  - Avendo modo di verificare quale delle due sia la chiave risultante,  $A$  può capire se il segreto di  $B$  è un numero pari o dispari, facendo perdere così un bit di sicurezza a  $B$ .
- Nel 2016 il 3% dei server HTTPS e il 34% dei server SSH accettava  $-1$  come valore di scambio.
- Questo principio può essere esteso a sottogruppi più grandi.
  - Attacco *Lim-Lee*.



# DH su campo primo: conclusioni

- La crittografia su curve ellittiche risulta migliore.
  - I maggiori browser hanno da tempo abbandonato l'utilizzo dello scambio di chiavi basato su Diffie-Hellman su campo primo.
- Diffie-Hellman non è sicuro per utilizzi post-quantistici.
  - L'*Algoritmo di Shor* sui computer quantistici permette di calcolare il logaritmo discreto in tempo polinomiale probabilistico.



## Parte 4

# Diffie-Hellman su curve ellittiche



# Crittografia su curve ellittiche: una panoramica

- La crittografia su curve ellittiche, a parità di sicurezza, richiede chiavi di molti meno bit.
  - Di conseguenza, le operazioni sono più veloci.
- Si comparano nella seguente tabella i bit necessari a parità di sicurezza nei tre protocolli descritti.

RSA e DH (bit del modulo)	ECC (bit dell'ordine)
1024	160
2048	224
3072	256
7680	384
15360	512





# Crittografia su curve ellittiche: il protocollo Diffie-Hellman

- Diffie-Hellman si presta molto bene all'applicazione su curve ellittiche.
- È stato adottato su larga scala negli ultimi 10 anni.
- Ad oggi è utilizzato da più del 65% degli scambi di chiave.



# DH su curve ellittiche: la teoria dietro al protocollo

- $A$  e  $B$ , che vogliono comunicare, si devono accordare su due parametri:
  - una curva ellittica prima  $E_p(a, b) = \{(x, y) \in \mathbb{Z}_p^2 \mid y^2 \bmod p = (x^3 + a \cdot x + b) \bmod p\} \cup \{0\}$ ;
  - un generatore  $G$  primo facente parte di un sottogruppo della curva di ordine  $n$ .
- Per lo scambio di chiave:
  - $A$  e  $B$  scelgono due interi casuali  $a, b$  tali che  $a, b < n$ ;
  - $A$  calcola  $P_A = a \cdot G$ ,  $B$  calcola  $P_B = b \cdot G$ ;
  - $A$  e  $B$  si scambiano rispettivamente  $P_A$  ed  $P_B$ ;
  - $A$  calcola  $k = a \cdot P_B$ ,  $B$  calcola  $k = b \cdot P_A$ ;
  - $k$  è condivisa poiché  $k = n_A \cdot n_B \cdot G$ .



# DH su curve ellittiche: uno sguardo alla sicurezza

- La sicurezza è basata sulla difficoltà di calcolare, dati due punti  $P$  e  $Q$ , il più piccolo intero  $k$  tale che  $P = k \cdot Q$ .
  - Chiamato *problema della risoluzione del logaritmo discreto su curve ellittiche*.
  - Il migliore attacco ad oggi è *Pollard rho* che ha complessità  $O(2^{b/2})$ .
  - Un attaccante per compromettere la comunicazione dovrebbe calcolare i più piccoli valori  $a$  oppure  $b$  tali che, rispettivamente:

$$P_A = a \cdot G \quad \text{oppure} \quad P_B = b \cdot G.$$

- Si fa affidamento a curve sicure, scelte da una lista del NIST.



# DH su curve ellittiche: i parametri della curva $P-384$

- È una delle curve ellittiche più utilizzate e raccomandate.
- $p = 2^{284} - 2^{128} - 2^{96} + 2^{32} - 1$ ;
- $a = -3$ ;
- $b = 27580193559959705877849011840389048093056905856361568521 \backslash$   
428707301988689241309860865136260764883745107765439761230575, avente la proprietà che  $2^{383} < b < 2^{384}$ ;
- $N \simeq 2^{384} - 2^{190}$ , che rappresenta l'*ordine della curva*, ovvero il numero di punti che ammette al suo interno.



# DH su curve ellittiche: attacco con curva invalida

- Attacco simile a quello descritto per DH su campo primo.
- Se il valore inviato non giace sulla curva ed ha un ordine basso  $q_i$ , un attaccante può calcolare il segreto della vittima modulo  $q_i$ .
- Se la vittima ricicla il suo segreto per molteplici connessioni, effettuando questo attacco più volte si può recuperare l'intero segreto.
- Si dovrebbe controllare che il punto inviato giaccia sulla curva corretta.
  - Nel 2015 tre di otto popolari librerie TLS non effettuavano questo controllo.
  - Si è stimato che nel 2015 lo 0,8% dei siti HTTPS non effettuavano questo controllo.



# DH su curve ellittiche: conclusioni

- La crittografia su curve ellittiche risulta ad oggi essere il miglior metodo per lo scambio di chiavi.
- Anche il protocollo Diffie-Hellman su curve ellittiche non è sicuro per utilizzi post-quantistici.
  - L'*Algoritmo di Shor* sui computer quantistici permette di calcolare il logaritmo discreto per le curve ellittiche in tempo polinomiale probabilistico.



# Parte 5

## Conclusioni



# Conclusioni

- La crittografia è in continuo studio ed evoluzione: ciò che oggi è considerato sicuro domani potrebbe non esserlo.
  - La fattorizzazione e il calcolo del logaritmo discreto sono problemi che ad oggi non conoscono algoritmi polinomiali, ma potrebbero esistere.
  - È importante e fondamentale restare aggiornati con gli studi.
- A luglio 2022 il NIST ha pubblicato gli algoritmi resistenti ai computer quantistici.
  - Un mese dopo questi algoritmi sono stati violati utilizzando dei classici calcolatori.





# Fine

Grazie per l'attenzione!





