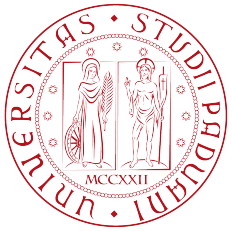


UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Matematica
" Tullio Levi-Civita "

Corso di Laurea Magistrale in Matematica

Possibile backdoor nella crittografia su curve ellittiche

Relatore:
Prof. Kloosterman Remke Nanne

Laureando:
Davide Motta
Matricola:
1206683

11 Dicembre 2020

Sai qual è la differenza tra uomo e donna?
Noi donne invecchiamo e voi uomini restate sempre bambini.

Anonimo

Ringraziamenti

Probabilmente questa è la parte più difficile dell'intera tesi. Chi ringraziare? Alla fine ho deciso che tutti quelli che in questi cinque anni hanno intersecato la loro vita con la mia meritano una menzione. Ovviamente è impossibile citarli uno ad uno per cui agli amici di sempre, agli isolani, ai "colleghi" universitari, a tutti i coinquilini, ai parenti, alla Murialdina, al mio relatore, a Padova dico GRAZIE.

Una menzione speciale è dovuta a mamma e papà che mi hanno permesso di raggiungere questo obiettivo e senza i quali non ce la avrei mai fatta.

Indice

Introduzione

A partire dalla seconda metà degli anni '90 prese piede lo sviluppo della *criptovirologia*, così definita dai pionieri Adam Young e Moti Yung. La criptovirologia studia come utilizzare la crittografia per implementare software e algoritmi fasulli, manomessi, il cui scopo principale è quello di far trapelare dati sensibili della vittima. Una sotto branca è la cosiddetta *cleptografia*, la quale si occupa di escogitare metodi per rubare in sicurezza e in anonimato le informazioni private di un utente. In questo contesto, Young e Yung scrissero una serie di articoli in cui veniva introdotto un attacco cleptografico chiamato *SETUP* [56]. In particolare, tale meccanismo è volto ad analizzare le fallacie dei sistemi a scatola nera, ovvero quei dispositivi in cui un utente ignora gli algoritmi interni, ma conosce solo input ed output. I due crittografi proposero molti attacchi SETUP implementati nei più famosi sistemi crittografici a chiave pubblica, dallo scambio di chiavi di Diffie-Hellman all'RSA, passando per il DSA e altri ancora.

Scopo principale di questa tesi è quello di fornire una panoramica generale, ma comunque esaustiva della cleptografia, specialmente su curve ellittiche, spiegata attraverso l'attacco SETUP. Per farlo ripercorreremo gli articoli nativi di Young e Yung, cercando di spiegare il funzionamento del SETUP nei sistemi di Diffie-Hellman, ElGamal e DSA. In realtà, sottolineeremo che tali attacchi non sono indipendenti tra di loro, ma presentano un comun denominatore: il *DLK*. Esso è sostanzialmente un schema generale che descrive un attacco cleptografico contro sistemi la cui sicurezza si basa sul problema del logaritmo discreto. Infine, cercheremo di andare oltre le basilari nozioni fornite da Young e Yung e vedremo come il DLK può essere adattato al contesto delle curve ellittiche (ECDLK). Quest'ultimo passo lo riteniamo molto importante, poiché, oggigiorno, i sistemi crittografici basati sulle curve ellittiche sono sempre più diffusi ed utilizzati. Diventa, quindi, importante saper riconoscere e neutralizzare eventuali manomissioni.

La tesi sarà strutturata in 5 capitoli.

Nel primo capitolo esporremo brevemente la teoria delle curve ellittiche privilegiando un approccio più pragmatico che teorico. Vederemo come l'insieme dei punti di una curva ellittica costituisca un gruppo abeliano additivo e forniremo qualche nozione e risultato di base come l'ordine del gruppo e di un punto. Infine, parleremo brevemente della complessità computazionale di algoritmi e vedremo come si particolarizza la questione per le curve ellittiche.

Nel capitolo 2 presenteremo le nozioni e gli esempi base presenti in qualunque libro di introduzione alla crittografia. Inizieremo fornendo qualche esempio storico come il cifrario di Cesare, per poi passare a descrivere sistemi più recenti, quali AES o DES. Passeremo, poi, alla narrazione della crittografia a chiave pubblica soffermandoci sui crittosistemi che avranno una importanza centrale nel prosegui della tesi. Stiamo parlando dello scambio di chiavi di Diffie-Hellman, del protocollo di cifratura e quello di firma digitale ideati

da ElGamal, e del DSA. Dopodiché, dedicheremo una sezione per scrivere le versioni ellittiche degli schemi appena citati. Quindi vedremo l'ECDH, gli schemi ECEE ed ECES ed infine il famoso ECDSA. Ogni descrizione sarà munita di algoritmo, un esempio con piccoli numeri e di un paragrafo inerente l'analisi della sicurezza. Infine, presenteremo un excursus atto a presentare i pro e i contro della crittografia classica e asimmetrica.

Nel terzo capitolo entreremo nel vivo della tesi. Infatti, presenteremo una panoramica generale della cleptografia e dei problemi relativi a dispositivi a scatola nera seguendo le idee di Young e Yung. Successivamente, ci immergeremo nella descrizione dell'attacco SETUP. Quindi, forniremo le definizioni principali che ci accompagneranno per tutto il resto dello scritto, ed in particolare analizzeremo un attacco chiamato Dicrete Log Kleptogram (DLK). Una volta esposto il DLK, ne presenteremo la trattazione della sicurezza. In seguito mostreremo come questo meccanismo possa essere inserito nei sistemi basati sul DLP. Per la precisione, disquisiremo riguardo l'attuazione di un attacco SETUP nei sistemi crittografici sopra citati (DH, ElGamal e DSA). Per tutti gli schemi, proporremo una algoritmo, un esempio con piccoli numeri e, ovviamente, l'analisi della sicurezza.

Nel capitolo 4 emuleremo quanto fatto nel precedente ma in chiave ellittica. Quindi, descriveremo il ECDLK sottolineando, per quanto possibile i vantaggi, e gli svantaggi rispetto al normale DLK. In seguito, descriveremo l'attacco SETUP negli schemi ECDH, ECEE, ECES ed ECDSA seguendo la stessa linea di pensiero usata nel capitolo 3. Tuttavia, cercheremo di capire quanto la realizzazione di un attacco SETUP possa essere davvero possibile nella vita reale. Per far ciò, analizzeremo le complessità computazionali degli algoritmi contaminati e vedremo di quanto si discostano dagli algoritmi sani. Tanto più è grande il divario, tanto più sarà facile rilevare la presenza di un meccanismo SETUP nel dispositivo mediante un attacco a cronometro. Infine, proveremo a fornire qualche idea per celare più efficacemente il SETUP.

Nel capitolo conclusivo riassumeremo quanto detto lungo la tesi, enfatizzando i risultati principali.

Capitolo 1

Introduzione alle Curve Ellittiche

Lo studio delle curve ellittiche non è di certo una attività recente, ma recenti sono le applicazioni in cui questa branca della geometria algebrica ha trovato spazio. Non è raro infatti, leggere libri che trattano di curve ellittiche con la consueta applicazione alla crittografia.

In questo primo capitolo introdurremo le curve ellittiche, fornendo le definizioni principali e fissando le notazioni che useremo in seguito. Anticipiamo fin da subito che prediligeremo la via più pragmatica per introdurre questi oggetti matematici, consci che sono possibili strade più astratte come fatto, per esempio, da Silverman in [43].

Mostreremo che l'insieme dei punti su una curva ellittica ha una naturale struttura di gruppo (abeliano) additivo, concetto importantissimo che determinò il definitivo successo in ambito applicativo di tali curve. Dopodiché, passeremo a descrivere le curve su campi finiti e definiremo l'ordine del gruppo e l'ordine di un punto, quest'ultimo serve a stabilire la natura del suddetto gruppo additivo. Infine, presenteremo due famiglie di curve ellittiche tutt'oggi utilizzate in applicazioni comuni.

1.1 Curve ellittiche nel piano affine

Nel corso di questa sezione lavoreremo nel piano affine $\mathbb{A}^2(\mathbb{K})$. Bisogna sottolineare che tratteremo le curve ellittiche come luogo degli zeri di una equazione cubica, tralasciando una descrizione generale e astratta.

Notazione 1. Nel seguito denoteremo con $\mathbb{K}$ un campo che potrà essere: $\mathbb{Q}, \mathbb{R}, \mathbb{C}, \mathbb{F}_q$ dove $q = p^k$ con $k \geq 1$ e $p \in \mathbb{N}$ un primo.

Diamo una prima definizione generale di curva ellittica.

Definizione 1.1. Sia $\mathbb{K}$ un campo. Una *curva ellittica* E su $\mathbb{K}$ è una curva regolare nella seguente forma (detta *equazione di Weierstrass generalizzata*)

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \qquad a_i \in \mathbb{K} \tag{1.1}$$

Precisamente, una curva ellittica E è l'insieme dei punti $(x, y) \in \mathbb{A}^2(\mathbb{K})$ che soddisfano l'equazione (1.1) più un punto, detto *punto all'infinito*, che denoteremo con $\mathcal{O}$.

Se $\mathbb{L}$ è un sottocampo di $\mathbb{K}$ allora denoteremo

$$E(\mathbb{L}) = \{\mathcal{O}\} \cup \{(x, y) \in \mathbb{L} \times \mathbb{L} \,:\, y^2 = x^3 + ax + b\}$$

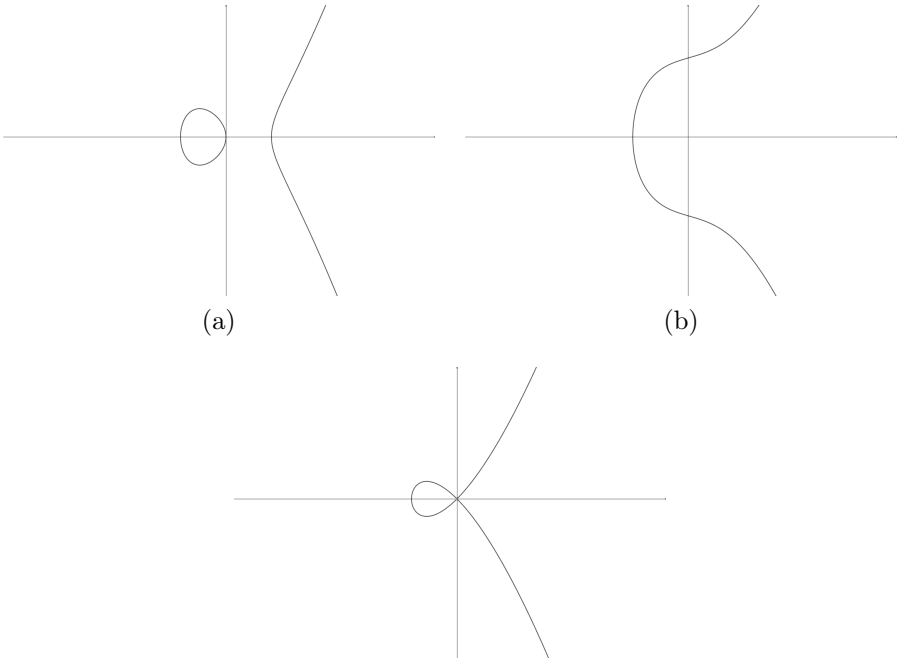


Figura 1.1: (a) e (b) sono curve ellittiche, (c) è una cubica con nodo, quindi non è una curva ellittica.

Se il campo sarà chiaro dal contesto, allora useremo solo la notazione E.

La forma dell'equazione (1.1) è notevolmente semplificabile se lavoriamo in campi con caratteristica diversa da 2 e 3. Infatti, consideriamo il cambio di coordinate

$$x \mapsto x - \frac{a_2}{3}$$
$$y \mapsto y - \frac{a_1 x + a_3}{2}$$

Una volta effettuata la sostituzione, l'equazione assume la forma più umana (detta *equazione di Weierstrass*)

$$y^2 = x^3 + ax + b \qquad a, b \in \mathbb{K}$$

Ricordiamo che, nella definizione sopra, si parla di curva regolare. L'ipotesi di regolarità di $y^2 = x^3 + ax + b$ si traduce nel richiedere che il polinomio $f(x) = x^3 + ax + b$ non abbia radici multiple. Infatti, se $f(x)$ non ha radici multiple, allora, da un noto teorema di algebra, $f'(x) = 3x^2 + a \neq 0$. D'altra parte, un punto (x, y) della curva è regolare se e solo se il gradiente di $F(x, y) = y^2 - x^3 - ax - b$ è sempre diverso da $(0, 0)$. Ma $\nabla F(x, y) = (-f'(x), 2y)$. Dunque la curva $y^2 = x^3 + ax + b$ è regolare se e solo se $x^3 + ax + b$ è privo di radici multiple. Questo equivale a dire che, lavorando con il discriminante,

$$\Delta(f) = -4a^3 - 27b^2 \neq 0$$

Nel seguito lavoreremo spesso con campi di caratteristica diversa da 2 e da 3. Il motivo non è prettamente estetico, ma anche computazionale. Infatti, in caratteristica 2

e 3, le curve ellittiche sono più vulnerabili ad attacchi basati sul problema del logaritmo discreto [15, 13, 18].

Quindi, possiamo dare la seguente definizione di curva ellittica.

Definizione 1.2. Sia $\mathbb{K}$ un campo con caratteristica diversa da 2 e 3. Sia $x^3 + ax + b$ un polinomio privo di radici multiple (ovvero $-4a^3 - 27b^2 \neq 0$), con $a, b \in \mathbb{K}$. Una *curva ellittica* E è l'insieme dei punti $(x, y) \in \mathbb{A}^2(\mathbb{K})$ che soddisfano l'equazione (detta *equazione di Weierstrass*)

$$y^2 = x^3 + ax + b \tag{1.2}$$

più un punto, detto *punto all'infinito*, che denoteremo con $\mathcal{O}$.

Come prima

$$E(\mathbb{K}) = \{\mathcal{O}\} \cup \{(x, y) \in \mathbb{K} \times \mathbb{K} \,:\, y^2 = x^3 + ax + b\}$$

Osservazione 1. Ovviamente si possono definire curve ellittiche anche in campi con caratteristica 2 e 3. Sempre con opportuni cambi di variabili, è possibile semplificare l'equazione (1.1) ed ottenere:

- Se $\text{char}(\mathbb{K}) = 2$, una curva ellittica è l'insieme dei punti che soddisfano a

  $$y^2 + cy = x^3 + ax + b$$

  oppure a

  $$y^2 + xy = x^3 + ax^2 + b$$

  più il punto all'infinito $\mathcal{O}$.

- Se $\text{char}(\mathbb{K}) = 3$, una curva ellittica è l'insieme dei punti che soddisfano a

  $$y^2 = x^3 + ax^2 + bx + c$$

  più il punto all'infinito $\mathcal{O}$.

1.2 Legge di gruppo

Consideriamo una curva ellittica E come in (1.2). La particolarità di tali curve è che è possibile dotare l'insieme $E(\mathbb{K})$ di una struttura algebrica, quella di gruppo. Dunque dobbiamo definire una operazione binaria associativa, trovare l'inverso e l'elemento neutro.

Definizione 1.3. Sia E una curva ellittica definita su $\mathbb{K}$ come in (1.2) e siano P e Q due punti sulla curva E. Definiamo l'operazione di somma $+ : E \times E \to E$, $(P, Q) \mapsto P + Q$ e l'inverso di P in accordo con le seguenti regole:

1. **Elemento neutro:** Se P è il punto all'infinito $\mathcal{O}$, allora definiamo $-P$ come $\mathcal{O}$. Per ogni altro punto Q definiamo $\mathcal{O} + Q = Q$.

2. **Elemento inverso:** Dato $P = (x, y)$, definiamo $-P := (x, -y)$.

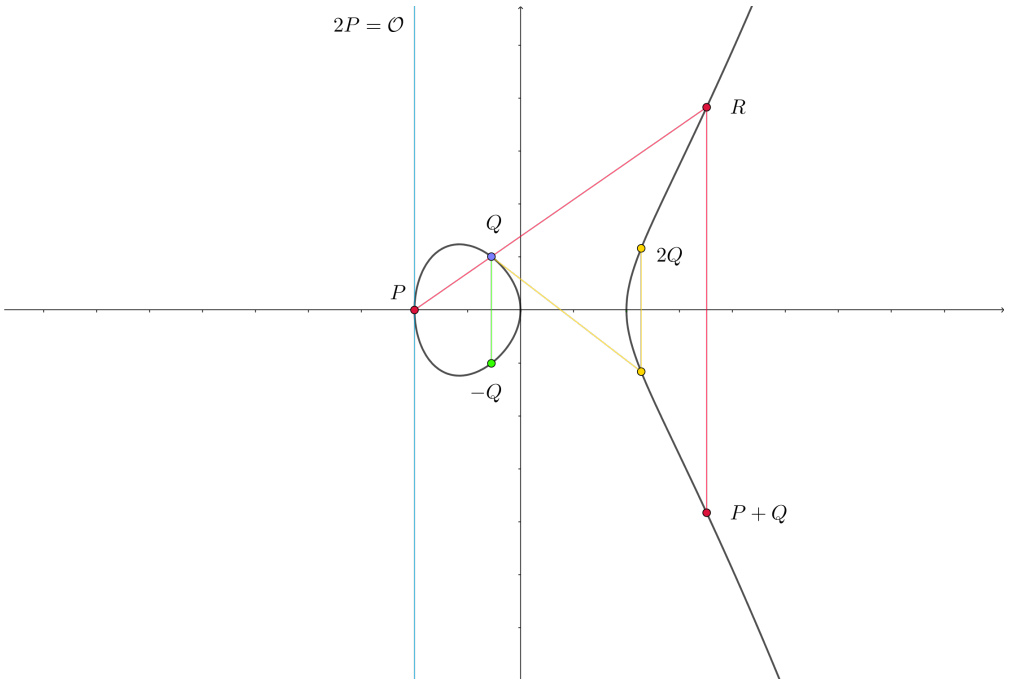


Figura 1.2: Operazioni tra punti della curva ellittica $y^2 = x^3 - x$.

3. $\boldsymbol{P+Q}$: Se $Q = -P$ allora definiamo $P+Q = \mathcal{O}$. Altrimenti, sia r la retta passante per i punti P e Q, allora r interseca la curva in un terzo punto R, a meno che r non sia la retta tangente alla curva in P, nel qual caso prendiamo $R = P$; se r è tangente in Q, allora $R = Q$. Dunque, definiamo $P + Q$ come $-R$.

4. $\boldsymbol{[2]P}$ Se $P = Q$, allora consideriamo la retta r tangente alla curva nel punto P. Sia R l'unico altro punto di r di intersezione con la curva. Definiamo $[2]P = P+Q = -R$ (R è $\mathcal{O}$ se la tangente è verticale). Questa operazione è definita come *doubling*.

Nella figura 1.2 è possibile apprezzare le operazioni tra punti appena definite.

Tale costruzione va giustificata. A tal fine, mostriamo che dati due punti P e Q con $Q \neq P, -P$, esiste sempre un terzo punto di intersezione con la curva E e la retta per P e Q. In realtà, l'esistenza ci è assicurata dal teorema di Bézout [8]. Tuttavia, presentiamo calcoli espliciti per la verifica, utili per poi determinare la complessità computazionale delle operazioni.

Siano $P = (x_1, y_1)$ e $Q = (x_2, y_2)$. Consideriamo la retta r passante per i punti P e Q. Ovvero $r : y = mx + q$. Allora

$$m = \frac{y_2 - y_2}{x_2 - x_1} \quad \text{e} \quad q = y_1 - mx_1$$

Ora, un punto di r, $(x, mx + q)$ appartiene alla curva E se e solo se

$$(mx + q)^2 = x^3 + ax + b$$

ovvero, se e solo se x è uno zero del polinomio $x^3 - m^2x^2 + (a - 2mq)x + (b - q^2)$. Ma noi sappiamo già che x_1 e x_2 sono zeri di tale polinomio per ipotesi. Inoltre la somma delle

radici del polinomio coincide con il coefficiente di x^2. Detta x_3 la terza radice, abbiamo che

$$x_1 + x_2 + x_3 = m^2$$

da cui

$$x_3 = m^2 - x_1 - x_2$$

Adesso è possibile scrivere il punto $P + Q = (x_3, y_3)$ in funzione di x_1, x_2, y_1, y_2.

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \tag{1.3}$$

$$y_3 = -y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) \quad \text{dove } y_3 = -(mx_3 + q) \tag{1.4}$$

Resta da trattare il caso $P = Q$. Ora m coincide con la derivata implicita dy/dx. Quindi

$$2y\frac{dy}{dx} = 3x^2 + a$$

da cui

$$m = \frac{dy}{dx} = \frac{3x_1^2 + a}{2y_1} \tag{1.5}$$

Se $y_1 = 0$ allora la retta tangente è verticale ($3x_1^2 + a \neq 0$ altrimenti il polinomio $x^3 + ax + b$ avrebbe radici multiple). Dunque si pone $[2]P = \mathcal{O}$.

Notiamo che, se P e Q sono punti con coordinate nel campo $\mathbb{K}$, allora anche $P + Q$ ha coordinate in $\mathbb{K}$. Dunque $E(\mathbb{K})$ è chiuso rispetto alla somma di punti appena definita.

Resta da dimostrare che la somma è associativa. Tuttavia, questo passaggio non è elementare e richiede conti pesanti e inutili per gli scopi di questa tesi.

Ricapitoliamo quanto detto in questa sezione nel seguente teorema.

Teorema 1. *Sia E una curva ellittica definita su un campo $\mathbb{K}$. Allora, la somma di punti $+ : E \times E \to E$, $(P, Q) \mapsto P + Q$ soddisfa le seguenti proprietà:*

1. ***Commutatività:*** *$P + Q = Q + P$ per ogni $P, Q \in E$.*

2. ***Esistenza dell'elemento neutro:*** *$P + \mathcal{O} = P$ per ogni $P \in E$.*

3. ***Esistenza dell'inverso:*** *Dato $P \in E$ esiste $Q \in E$ tale che $P + Q = \mathcal{O}$.*

4. ***Associatività:*** *$(P + Q) + R = P + (Q + R)$ per ogni $P, Q, R \in E$.*

In altre parole, i punti di E formano una gruppo abeliano additivo avente $\mathcal{O}$ come elemento neutro.

Dimostrazione. Una dimostrazione dell'associatività è presente in [49]. □

1.3 Curve ellittiche nel piano proiettivo

In realtà, lo spazio naturale dove lavorare con le curve ellittiche è il piano proiettivo $\mathbb{P}^2(\mathbb{K})$. Qui, il punto all'infinito assume solidità fisica e le operazioni tra i punti divengono più semplici da trattare. In effetti, scompaiono le inversioni, che sono solitamente le operazioni più dispendiose, in ambito computazionale, svolte da un computer.

Nel seguito denoteremo con $[x, y, z]$ i punti del piano proiettivo.

Le curve ellittiche in $\mathbb{P}^2(\mathbb{K})$ sorgono dal naturale processo di omogenizzazione. Diamo dunque la seguente definizione per curve ellittiche nella forma di Weierstrass.

Definizione 1.4. Sia $\mathbb{K}$ un campo con caratteristica diversa da 2 e 3. Sia $x^3 + ax + b$ un polinomio privo di radici multiple, con $a, b \in \mathbb{K}$. Una *curva ellittica* E è l'insieme dei punti $[x, y, z] \in \mathbb{P}^2(\mathbb{K})$ che soddisfano l'equazione

$$y^2 z = x^3 + axz^2 + bz^3 \tag{1.6}$$

Con questa descrizione, il punto all'infinito $\mathcal{O}$ è implicito tra le soluzioni dell'equazione, in particolare si ha (per questa forma) $\mathcal{O} = [0, 1, 0]$. Tale punto si può pensare graficamente come il punto all'infinito dell'asse y (sia sopra che sotto). Inoltre $\mathcal{O}$ è l'unico punto non affine ed è regolare. Infatti, detta $F(x, y, z) = y^2 z - x^3 - axz^2 - bz^3$. Allora $\nabla F(\mathcal{O}) = (0, 0, 1)$ (gli altri punti sappiamo essere regolari per ipotesi). Per quanto detto nella sezione 1.1, la curva (1.6) è regolare.

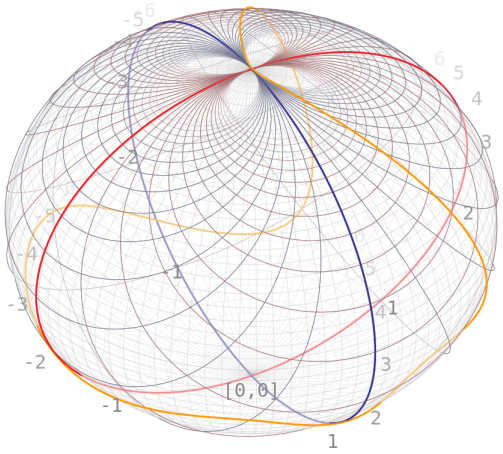


Figura 1.3: Una rappresentazione in $\mathbb{P}^2(\mathbb{K})$ della curva $y^2 = x^3 - 2x + 2$ (in giallo). Notare il passaggio per il punto $\mathcal{O}$ (intersezione assi).

1.4 Curve ellittiche su campi finiti

In questa sezione presenteremo qualche risultato riguardante curve ellittiche definite in campi finiti $\mathbb{F}_q$ con $q = p^k$, dove $p, k \in \mathbb{N}$ e p è un numero primo diverso da 2. Solitamente, nelle applicazioni è raro usare campi il cui ordine è la potenza di un primo dispari, quindi,

nella maggior parte dei casi, si può assumere senza perdita di generalità che $q = p$ o $q = 2^k$ ma quest'ultimo, come già detto, non lo considereremo.

L'interesse per lo studio di questi oggetti nasce dalla possibilità di costruire un sistema crittografico, basato sulla struttura di gruppo definita nella sezione 1.2.

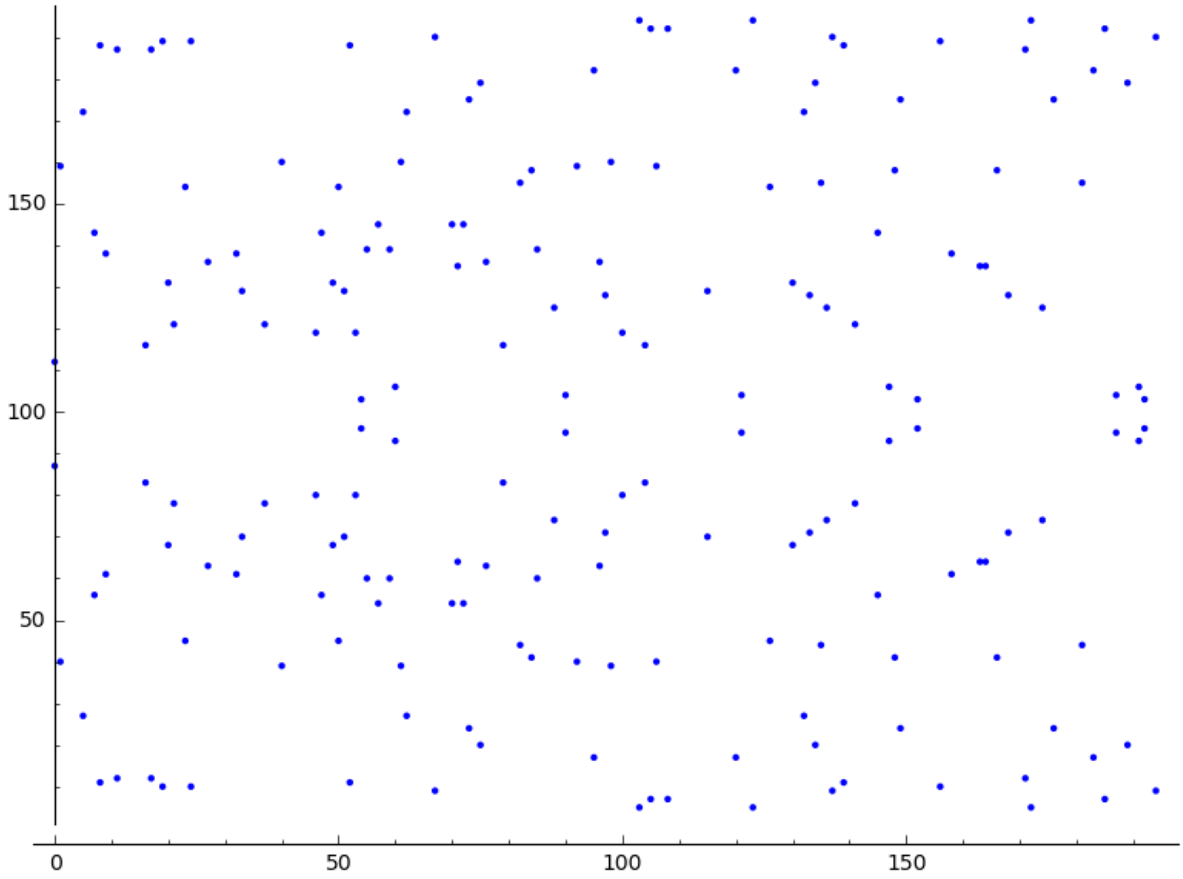


Figura 1.4: Curva ellittica $y^2 = x^3 + 7$ in $\mathbb{F}_{199}$

1.4.1 Ordine del gruppo $E(\mathbb{F}_q)$

Un primo passo interessante è quello di contare di punti di una curva ellittica e, di conseguenza, calcolare l'ordine del gruppo $E(\mathbb{F}_q)$.

Notazione 2. Nel seguito denoteremo $\#E(\mathbb{F}_q) = |E(\mathbb{F}_q)|$.

Un risultato di straordinaria importanza è il seguente teorema, congetturato da Artin ma dimostrato da Hasse, da cui prese il nome.

Teorema 2 (Hasse)**.** *Sia E una curva ellittica definita nel campo $\mathbb{F}_q$. Allora*

$$|\#E(\mathbb{F}_q) - (q+1)| \leq 2\sqrt{q} \tag{1.7}$$

Dimostrazione. Vedere [43], oppure [49]. □

Questo teorema, fornisce un limite superiore ed inferiore per $\#E(\mathbb{F}_q)$, cosa a priori assolutamente non scontata. Tuttavia, non fornisce alcun algoritmo per calcolare i punti di E, anche se possiamo comprendere che, a livello teorico, calcolare $\#E(\mathbb{F}_q)$ non è molto difficile.

Esempio 1. Sia E la curva $y^2 = x^3 + 1$ definita nel campo $\mathbb{F}_5$. Un modo per contare i punti è quello di trascrivere tutti i possibili valori per x. Dopodiché, si trova $x^3+1 \bmod 5$ e si considerano solo quei valori che sono quadrati modulo 5. Il risultato finale è la tabella sottostante.

x	x^3+1	y	Punti
0	1	± 1	$(0,1),(0,4)$
1	2	-	-
2	4	± 2	$(2,2),(2,3)$
3	3	-	-
4	0	0	$(4,0)$
$\mathcal{O}$		$\mathcal{O}$	$\mathcal{O}$

Dunque $\#E(\mathbb{F}_q) = 6$.

Dall'esempio appena presentato, si evince che il conteggio dei punti si basa sul sapere se $x^3 + ax + b$ è un quadrato in $\mathbb{F}_q$. Tale procedimento si può riassumere con il simbolo di Legendre per campi $\mathbb{F}_p$ con p primo . Tale simbolo può essere generalizzato a campi finiti qualunque $\mathbb{F}_q$.

Definizione 1.5. Sia p un numero primo dispari ed x un numero intero. Si definisce il *simbolo di Legendre* come:

$$\left(\frac{x}{p}\right) = \begin{cases} 1 & \text{se } x \text{ è un quadrato modulo } p \\ -1 & \text{se } x \text{ non è un quadrato modulo } p \\ 0 & \text{se } p \mid x \end{cases}$$

Sia $\mathbb{F}_q$ con q dispari. Sia $x \in \mathbb{F}_q$. Si definisce il *simbolo di Legendre generalizzato* come:

$$\left(\frac{x}{\mathbb{F}_q}\right) = \begin{cases} 1 & \text{se } x \text{ è un quadrato in } \mathbb{F}_q^\times \\ -1 & \text{se } x \text{ non è un quadrato in } \mathbb{F}_q \\ 0 & \text{se } x = 0 \end{cases}$$

Con questa definizione, il seguente teorema è di immediata dimostrazione.

Teorema 3. *Sia E una curva ellittica di equazione $y^2 = x^3 + ax + b$ in $\mathbb{F}_q$. Allora*

$$\#E(\mathbb{F}_q) = q + 1 + \sum_{x \in F_q} \left(\frac{x^3 + ax + b}{\mathbb{F}_q}\right) \tag{1.8}$$

Dimostrazione. Preso un $x \in \mathbb{F}_q$, allora l'equazione $y^2 = x^3 + ax + b$ ha esattamente $1 + \left(\frac{x^3+ax+b}{\mathbb{F}_q}\right)$ soluzioni. Facendo questo ragionamento per tutti gli elementi di $\mathbb{F}_q$ e aggiungendo un $+1$ per il punto all'infinito $\mathcal{O}$, si arriva a

$$\#E(\mathbb{F}_q) = 1 + \sum_{x \in F_q} \left(1 + \left(\frac{x^3+ax+b}{\mathbb{F}_q}\right)\right)$$

da cui la tesi. □

In termini pratici, l'equazione (1.8) non è assolutamente efficiete per conteggiare i punti di una curva ellittica. Infatti, il simbolo di Legendre può essere calcolato in $O(\log^3 q)$ passi [24]. Quindi, la formula (1.8) richiede un tempo $O(q \log^3 q)$. Tuttavia, R. Schoof ideò un algoritmo polinomiale per il calcolo di $\#E(\mathbb{F}_q)$ che si basa sul teorema 2. Per una descrizione di tale algoritmo, si veda [39].

1.4.2 Ordine di un punto

L'ordine di un punto P è un concetto del tutto analogo all'ordine di un elemento $g \in \mathbb{Z}_p$.

Definizione 1.6. Sia E una curva ellittica definita in un campo finito $\mathbb{F}_q$. Sia $P \in E(\mathbb{F}_q)$. L' *ordine* di P è il più piccolo intero positivo k tale che

$$[k]P = \mathcal{O}$$

Denoteremo l'ordine di P con $ord(P)$.

Dunque, un punto P genera un sottogruppo di $E(\mathbb{F}_q)$ di ordine $k = ord(P)$. Quindi, per il teorema di Lagrange, l'ordine di un punto divide l'ordine di $E(\mathbb{F}_q)$.

Notiamo che l'ordine di un punto può essere utilizzato per calcolare $\#E(\mathbb{F}_q)$. Infatti, dal teorema 2 sappiamo che $q+1-2\sqrt{q} < \#E(\mathbb{F}_q) < q+1+2\sqrt{q}$, quindi, se $ord(P)$ è più grande di $4\sqrt{q}$ allora ci può essere un unico multiplo di $ord(P)$ nell'intervallo di $\#E(\mathbb{F}_q)$. Tale multiplo sarà proprio $\#E(\mathbb{F}_q)$.

Esempio 2. Consideriamo la curva $E : y^2 = x^3 + 7x + 1$ definita in $\mathbb{F}_{101}$. Diamo per buono che l'ordine di $P = (0, 1)$ sia 116. Siccome $4\sqrt{101} < 50$, allora $0 < 116$, quindi $\#E(\mathbb{F}_q)$ è un multiplo di 116 per quanto detto sopra. Il teorema di Hasse afferma che

$$101 + 1 - 2\sqrt{101} \leq \#E(\mathbb{F}_q) \leq 101 + 1 + 2\sqrt{101}$$

ovvero, arrotondando

$$82 \leq \#E(\mathbb{F}_q) \leq 122$$

Dunque, l'unico multiplo di 116 in questo intervallo è 116 stesso, da cui $\#E(\mathbb{F}_q) = 116$.

Per calcolare l'ordine di un punto esistono vari algoritmi, uno di questi è il *Baby-step, Giant-step* di cui si può trovare una descrizione in [49].

Conoscere l'ordine di un punto può essere utile anche per determinare la struttura del gruppo $E(\mathbb{F}_q)$. Infatti, vale il seguente teorema, la cui dimostrazione si basa sugli n gruppi di torsione, ovvero gli insiemi dei punti P, tali che $[n]P = \mathcal{O}$.

Teorema 4. *Sia E una curva ellittica definita sul campo $\mathbb{F}_q$. Allora*

$$E(\mathbb{F}_q) \cong \mathbb{Z}_n \quad o \quad E(\mathbb{F}_q) \cong \mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2}$$

per un qualche intero $n \geq 1$, oppure $n_1, n_2 > 1$ con $n_1 \mid n_2$.

Esempio 3. Abbiamo mostrato nell'Esempio 1 che il gruppo $E(\mathbb{F}_5)$, dove $E : y^2 = x^3+1$, ha ordine 6. Siccome non è possibile scrivere 6 come prodotto di due interi di cui uno divide l'altro, possiamo concludere che

$$E(\mathbb{F}_5) \cong \mathbb{Z}_6$$

ed un generatore è $(2, 3)$ per esempio.

1.4.3 Curve ellittiche nella storia

Vale la pena notare che esistono famiglie di curve ellittiche che hanno una certa rilevanza in ambito applicativo, tanto da meritarsi un nome. Presentiamo brevemente due di queste famiglie: le *curve di Montgomery* e le *curve di Edwards*.

Montgomery Il nome di questa famiglia si deve a Peter L. Montgomery che per primo le studiò nel 1987 [30].

Definizione 1.7. Sia $\mathbb{K}$ un campo di caratteristica diversa da 2. Una *curva di Montgomery* è una curva ellittica nella forma

$$by^2 = x^3 + ax^2 + x \tag{1.9}$$

con $a, b \in \mathbb{K}$ e $b(a^2 - 4) \neq 0$.

Precisiamo che queste curve sono equivalenti alla forma di Weierstrass (1.2) tramite cambi di coordinate.

Una rappresentate degno di nota all'interno di questa famiglia è la Curve25519 studiata da Bernstein nel non lontano 2006. La sua scrittura è

$$y^2 = x^3 + 486662x^2 + x$$

e, ad oggi, è largamente utilizzata in ambito crittografico grazie alla velocità e sicurezza che fornisce per lo scambio di chiavi di Diffie-Hellman [4]. In [47] è disponibile una lista di applicazioni della Curve25519, tra queste, spiccano i nomi di Telegram e WhatsApp.

Edwards Nel 2007 Harold Edwards si cimentò nell'analisi di questa famiglia, intuendone la portata applicativa [16].

Definizione 1.8. Sia $\mathbb{K}$ un campo di caratteristica diversa da 2. Una *curva di Edwards* è una curva ellittica nella forma

$$x^2 + y^2 = 1 + dx^2y^2 \tag{1.10}$$

con $d \in \mathbb{K} \setminus \{0, 1\}$.

Anche questa famiglia è in certo senso equivalente alle classica forma di Weierstrass (equivalenza birazionale).

Bernstein e Lange, intuirono le potenzialità di queste curve. Infatti, esse permettono un calcolo molto più rapido per la somma di punti [6].

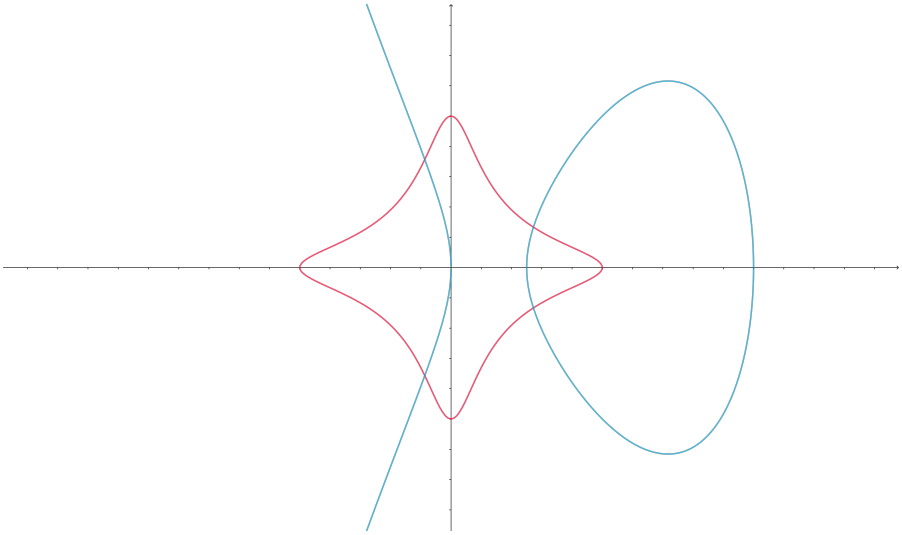


Figura 1.5: In rosso una curva di Edwards, in blu una di Montgomery.

1.5 Complessità computazionale

Quando si parla di efficienza di un algoritmo sono due i fattori fondamentali da tener presente: il costo temporale ed il costo spaziale. Insieme formano la complessità computazionale.

Il costo temporale si riferisce alla velocità di esecuzione di un algoritmo in termini di secondi è sarà l'oggetto preso in analisi di questa sezione. Il secondo concerne la quantità di risorse necessarie all'algoritmo per l'elaborazione. È solitamente misurata in termini di byte di memoria necessari per memorizzare le informazioni temporanee nel corso dell'esecuzione dell'algoritmo. Nel seguito confonderemo per motivi di comprensione costo temporale con l'intera complessità computazionale.

Dato un algoritmo, vorremmo poter stimare il tempo di elaborazione necessario per giungere alla soluzione del problema. In tal senso l'unità di misura è *l'operazione elementare* (o *bit operation*) che è l'operazione che, dati tre bit, ne fornisce la somma.

Una volta definita l'unità di misura, ci chiediamo quando un algoritmo è efficiente, cioè la sua esecuzione è rapida, veloce. In tal senso, parleremo di *complessità polinomiale*.

Definizione 1.9. Un algoritmo ha *complessità polinomiale* se esiste una costante $c > 0$ tale che il suo costo sul numero m di n bit è $O(n^C) = O(\log^C m)$ operazioni elementari.

Al contrario, quando un algoritmo è inefficiente, si parlerà di *complessità esponenziale*.

Definizione 1.10. Diremo che un algoritmo possiede *complessità esponenziale* se esiste una costante $C > 0$ tale che il suo costo sul numero m di n bit è $O(e^{Cn}) = O(m^C)$ operazioni elementari.

Per esempio, lavorando in campi finiti $\mathbb{F}_p$, il costo dell'addizione è $O(\log p)$ quindi rientra nella categoria della complessità polinomiale. La moltiplicazione costa alla peggio $O(\log^2 p)$, ma esistono algoritmi molto più efficienti che svolgono il prodotto con complessità $O(\log p \log\log p \log\log\log p)$. Infine, il costo di una potenza modulare $a^k \bmod p$ è pari a $O(\log^2 p \log k)$ o $O(\log^3 p)$ se $k < p$ alla peggio.

In realtà, esistono algoritmi specifici per la moltiplicazione che funzionano più o meno bene a seconda del campo in cui si sta lavorando. Per tale motivo, denoteremo con $\mathbf{M}$ il costo di una moltiplicazione, intendedno che esso denoterà sempre il costo del miglior algoritmo per la moltiplicazione in un dato campo. Quindi, per esempio, il costo di una potenza modulare sarà denotato con $O(\mathbf{M}\log k)$. Per ulteriori dettagli si può consultare [12].

Tuttavia, queste appena descritte sono solo stime che descrivono l'andamento generale dell'esecuzione di un algoritmo, e servono a stabilire quale algoritmo sia più efficiente rispetto ad altri. Chiaramente, anche un algoritmo polinomiale potrebbe risolvere un dato problema in anni se non in decadi. Per tale motivo, è buona cosa descrivere gli algoritmi in termini delle operazione più dispendiose ivi presenti. Nell'aritmetica sui campi finiti esse sono la già citata moltiplicazione, che denoteremo con $\mathbf{M}$, l'elevamento al quadrato $\mathbf{S}$ (squaring) e l'inversione $\mathbf{I}$, questa è di gran lunga la più costosa delle tre.

Nel contesto delle curve ellittiche, la particolarizzazione sopra proposta è molto importante e ne vedremo un esempio concreto tra poco. Prima però, sfruttiamo le equazioni (1.3), (1.4) e (1.5) per calcolare il costo di un doubling $\mathbf{D}$ e della somma di due punti $\mathbf{A}$. I risultati sono portati nella tabella qui sotto.

Operazione	Costo
$\mathbf{A}$: $P+Q$	$1\mathbf{I}$, $2\mathbf{M}$, $1\mathbf{S}$
$\mathbf{D}$: $2P$	$1\mathbf{I}$, $2\mathbf{M}$, $2\mathbf{S}$

Tabella 1.1: Costi di somma e doubling

Moltiplicazione scalare sulle curve ellittiche

La potenza modulare su campi finiti è l'operazione più dispendiosa all'interno di un algoritmo e ne determina pesantemente il costo temporale. In ambito ellittico, l'operazione analoga è il prodotto punto per scalare $[k]P$ (detto semplicemente *prodotto scalare*), per tale motivo vorremo conoscerne la complessità computazionale e, data la somiglianza, ci aspettiamo sia analoga a quella della potenza modulare, cioè $O(\mathbf{M}\log k)$.

Lemma 1. *Sia E una curva ellittica come in (1.2) definita su un campo finito $\mathbb{F}_q$ con caratteristica diversa da 2 e da 3. Dato un punto $P \in E(\mathbb{F}_q)$, le coordinate di $[2]P$ possono essere calcolate in $O(\boldsymbol{M})$ operazioni elementari.*

Dimostrazione. Analizzando le equazioni (1.3) e (1.4), notiamo che sono presenti al più una dozzina di operazioni (elevamento al quadrato, moltiplicazioni, divisioni, sottrazioni, addizioni). Tutte queste operazioni costano al più $O(\mathbf{M})$ singolarmente (la moltiplicazione e la divisione sono le più dispendiose). Tuttavia, per proprietà di O-grande, il costo totale del doubling è

$$O(12 \cdot \mathbf{M}) = O(\mathbf{M})$$

□

Proposizione 1. *Sia E una curva ellittica come in (1.2) definita su un campo finito $\mathbb{F}_q$ con caratteristica diversa da 2 e da 3. Dato un punto $P \in E(\mathbb{F}_q)$, le coordinate di $[k]P$ possono essere calcolate in $O(\boldsymbol{M}\log k)$ operazioni elementari.*

Dimostrazione. L'idea si basa sull'analogo ellittico del metodo *square-and-multiply.* Infatti, per calcolare $[k]P$, possiamo fare

$$2P, 4P, 8P \dots$$

seguendo l'algoritmo qui proposto.

Algoritmo 1: Calcolo di $[k]P$

input : E, q, P, k
output: $R = [k]P$
$k = k_0 + k_1 \cdot 2 + k_2 \cdot 2^2 + \cdots k_l 2^l, \quad k_i \in \{0, 1\}$
$R \leftarrow \mathcal{O}$
$A \leftarrow P$
for $i = 0, \dots, l-1$ **do**
  if $k_i = 1$ **then**
    $R \leftarrow R + A$
  end
  $A \leftarrow 2A$
end
return R

Dunque, l'algoritmo svolge in totale $l = O(\log k)$ passi. Ogni passo consta di un doubling, il cui costo è $O(\mathbf{M})$ per lemma 1. Dunque, in totale, l'algoritmo calcola $[k]P$ in $O(\mathbf{M} \log k)$ operazioni elementari. □

Proposizione 2. *Sia E una curva ellittica come in* (1.2) *definita su un campo finito $\mathbb{F}_q$ con caratteristica diversa da 2 e da 3. Dato un punto $P \in E(\mathbb{F}_q)$, le coordinate di $[k]P$ possono essere calcolate in non più di $\log_2 k$ doubling e $\log_2 k$ addizioni di punti.*

Dimostrazione. Dall'espressione binaria di k deduciamo che $2^l \leq k < 2^{l+1}$, quindi, $l \leq \log_2 k$ il quale è anche la lunghezza del loop presente in 1. □

Quindi, l'algoritmo dipende fortemente dal numero di 1 presenti nell'espansione binaria di k. In media ce ne sono la metà, ovvero $l/2 \approx t/2$ dove t denota i numero di bit di q (nelle applicazioni k e q hanno lo stesso numero di bit). Quindi, il tempo medio di esecuzione dell'algoritmo è circa pari a quello di $t/2$ somme di punti e t doubling. Quindi, in totale abbiamo

$$\frac{t}{2}\mathbf{A} + t\mathbf{D}$$

ovvero, in termini delle operazioni di campo **M**, **S**, **I**

$$\frac{3}{2}t\mathbf{I} + 3t\mathbf{M} + \frac{5}{2}\mathbf{S} \tag{1.11}$$

dove quest'ultima relazione segue dalla tabella 1.1

La proposizione 1 sembra suggerire che moltiplicazione scalare su curve ellittiche ed elevamento alla potenza su campi finiti performino con le stesse tempistiche. Ciò non è vero; infatti, supponendo $q = p$, il calcolo di $[k]P$ è notevolmente più lento rispetto ad

$a^k \bmod p$. Questo è dovuto principalmente alla formula (1.11) dove compaiono addirittura delle inversioni di campo, le quali non sono presenti nell' algoritmo dello square-and-multiply sui campi finiti (esso utilizza al più t moltiplicazioni e circa $t/2$ elevamenti al quadrato).

Capitolo 2

Introduzione alla crittografia

Per introdurre in modo corretto i concetti della crittografia bisognerebbe spendere parole su parole, tanto è vasto e complesso questo mondo. Tuttavia, compito di questo capitolo non è tanto quello di sviscerare i meandri bui e nascosti dell'arte crittografica, quanto quello di iniziare eventuali neofiti a questa disciplina. Sarà preferito, quindi, un approccio più esemplificativo che teorico, in modo da prediligere l'intuizione alla ragione.

Pertanto, in questo primo capitolo introdurremo gli oggetti necessari per definire un crittosistema. Successivamente, descriveremo degli esempi classici di sistemi crittografici per poi introdurre gli schemi principali che saranno al centro della trattazione di questa tesi: lo scambio di chiavi di Diffie-Hellman (DH), i sistemi di ElGamal e il DSA. Finiremo, quindi, descrivendo le varianti ellittiche degli appena citati protocolli. Baseremo le nostre parole soprattutto sui testi [24] e [33].

In tutti questi esempi, faremo uso di tre famosi attori, Alice, Bob ed Eve. I primi due giovani saranno intenti a comunicare tra di loro, mentre Eve cercherà di intercettare e decifrare i messaggi. Questi tre personaggi sono la personificazione di un generico utente, una server, una banca, un sito ecc. (Alice e Bob), e di un hacker (Eve) e serviranno per semplificare i concetti esposti.

2.1 Crittositemi

La crittografia si occupa di consentire la trasmissione di un messaggio in modo tale da celarne il contenuto ad eventuali terze parti. Inoltre, solo il destinatario deve possedere gli strumenti necessari per poter leggere il contenuto.

Dunque, seguendo le definizioni proposte in [24], per comunicare c'è bisogno anzitutto di un *alfabeto* $\mathcal{A}$. Tale insieme di caratteri può essere pensato come il consueto alfabeto scolastico, ma nelle applicazioni digitali $\mathcal{A} = \mathbb{Z}_N$ dove $N \approx 2^{512}$. Avendo un alfabeto, possiamo costruire i *messaggi in chiaro*, che raggruppiamo nell'insieme $\mathcal{M} = \mathcal{A}^n$. Accanto a questo insieme, ci sarà il corrispettivo dei *messaggi cifrati* $\mathcal{C}$. A questo punto, non resta che scegliere una *funzione crittografica* $f : \mathcal{M} \to \mathcal{C}$ che, dato un messaggio in chiaro, lo cifra secondo l'algoritmo scelto (di per sè, f è detta *funzione di cifratura*). La funzione inversa f^{-1} è detta di *decifratura* e il nome lascia trapelare la sua funzionalità. Tali mappe, all'interno dello stesso sistema, dipenderanno da dei parametri dette *chiavi* (di cifratura per f e decifratura per f^{-1}).

Abbiamo collezionato tutti gli ingredienti e possiamo definire un sistema crittografico, detto anche *crittositema*.

Definizione 2.1. Un *crittositema* è una quaterna $(\mathcal{M}, \mathcal{C}, f, f^{-1})$.

Un crittositema, deve rispettare i seguenti requisiti.

- *Riservatezza:* i messaggi inviati devono poter essere accessibili ai soli destinatari autorizzati.
- *Integrità:* non ci devono essere manomissioni dei messaggi inviati.
- *Autenticità:* il destinatario deve essere certo della provenienza del messaggio.
- *Non ripudiabilità:* il mittente non deve poter disconoscere di aver spedito il messaggio.

Vale la pena dire fin da subito che la crittografia si può dividere in due grandi filoni narrativi, quello classico e quello moderno. Tale suddivisione si basa sull'utilizzo delle chiavi di cifratura e decifratura.

Definizione 2.2. Un crittosistema si dice *a chiave simmetrica* (o *classico*) se la funzione di decifratura coincide, o può essere dedotta facilmente da quella di cifratura.

Definizione 2.3. Un crittosistema si dice *a chiave pubblica* (o *asimmetrico*) se, conoscendo la funzione f, non è possibile risalire ad f^{-1} celermente, cioè senza compiere calcoli di lunghezza proibitiva.

Nelle pagine seguenti forniremo alcuni esempi tradizionali di crittositemi, evidenziando l'eventuale appartenenza ad uno dei due gruppi appena definiti.

2.2 Crittografia classica

Che cosa ha spinto l'uomo a sviluppare la crittografia? L'esigenza di trasmettere messaggi segreti, leggibili solo al mittente e al ricevente . A domanda complessa, risposta semplice. Quindi, non deve stupire che tecniche e strumenti crittografici siano stati scoperti più di 4500 anni fa, analizzando geroglifici egiziani. Gli Egiziani inventarono la *scitala*, una bacchetta di legno attorno alla quale veniva avvolta un strisciolina di pelle contenete il messaggio. Quando la striscia era srotolata, il testo non era intellegibile. Ma chi possedeva la bacchetta dal giusto diametro poteva decifrare il messaggio. Questo è un primo esempio di sistema crittografico a chiave simmetrica: la chiave (bacchetta di legno) è la stessa per cifrare e decifrare il messaggio.

Figura 2.1: La scitala

2.2.1 Il cifrario di Cesare

Anche i Romani possedevano buone conoscenze crittografiche. In tale periodo, gran fortuna ebbe il cifrario di Cesare, uno dei più antichi algoritmi crittografici mai scoperti.

Così, lo storico Svetonio, nel "*De Vita Cesarum*", spiega come Giulio Cesare utilizzava il sistema crittografico che porta il suo nome.

> Extant et ad Ciceronem, item ad familiares domesticis de rebus, in quibus, si qua occultius perferenda erant, per notas scripsit, id est sic structo litterarum ordine, ut nullum verbum effici posset: quae si qui investigare et persequi velit, quartam elementorum litteram, id est D pro A et perinde reliquas commutet.

Non è difficile capire come si utilizza il cifrario di Cesare. Consideriamo l'alfabeto latino di 26 caratteri e numeriamoli in ordine da 0 a 25. Alice e Bob fissano un numero da 0 a 25 che sarà la chiave segreta K (dal testo sopra si evince che Cesare usava $K = 3$). Per cifrare basta sommare K ad ogni carattere del testo in chiaro, ricominciando il giro dalla "A" se si eccede oltre il 25 (la "Z"), ovvero effettuando calcoli modulo 26. L'operazione di decifratura consiste, invece, nel sottrarre K. Da qui si deduce che il cifrario è a chiave simmetrica.

Oggigiorno, un sistema crittografico di questo tipo è assai obsoleto. Infatti, esso è vulnerabile ad un attacco del tipo "forza bruta", cioè ricerca esaustiva della chiave K. Con solo 26 chiavi da verificare, non è difficile decifrare il messaggio. Con l'avvento dell'analisi delle frequenze (XI secolo) poi, tale cifrario è caduto in disuso. In effetti, lettere ripetute consecutivamente indicano la presenza di doppie. Nella lingua italiana, la lettera finale è spesso una vocale; inoltre, le lettere "a","e","i" compaiono con maggior frequenza in un testo. Con queste ed altre considerazioni si possono rompere cifrari simili a quello di Cesare.

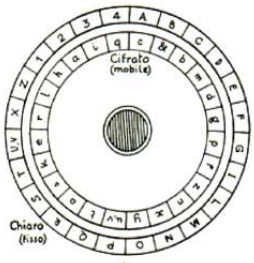


Figura 2.2: Tavola utilizzata per il cifrario di Cesare

2.2.2 Il codice di Vernam

Immaginiamo di vivere in un periodo politicamente delicato come quello della Guerra Fredda. Immaginiamo di essere il governo degli Stati Uniti d'America e di voler aprire un canale di comunicazione sicuro con l'URSS. Per far ciò vorremmo poter cifrare i nostri messaggi con un sistema crittografico *perfetto*, ovvero tale che, l'unica informazione in possesso di un eventuale intruso è la lunghezza del messaggio. In questo contesto entra in scena il cifrario di Vernam, il quale fu effettivamente usato nelle comunicazioni tra Washington e Mosca negli anni Sessanta. La cosiddetta *linea rossa*.

Testo in chiaro:	S	E	G	R	E	T	O
Chiave casuale:	C	A	I	L	N	B	R
Testo cifrato:	U	E	O	C	R	U	F

Tabella 2.1: Esempio di applicazione del cifrario di Vernam

Come nel cifrario di Cesare, i caratteri alfabetici sono numerati da 0 a 25. Dopodiché, Alice e Bob generano una chiave *casuale* lunga almeno quanto il testo in chiaro. Per cifrare, Alice codifica ogni lettera alla Cesare, ovvero trasla il primo carattere tanto quanto è indicato dalla prima lettera della chiave, e via così (vedere Tabella 2.1) . La decodifica avviene in modo ovvio. Alla fine del processo, Alice e Bob distruggono la chiave.

A livello concreto, il codice Vernam prevede l'edizione tipografica di blocchi cartacei uguali, come i calendari a strappo, con un foglio per ogni giorno, sui quali sono stampate lunghe sequenze di caratteri casuali. Alice e Bob possiedono ciascuno lo stesso blocco con le chiavi da usare giorno per giorno.

Questo schema, teoricamente inattaccabile, mostra pesanti limiti realizzativi. Uno su tutti è la difficoltà di generare sequenze casuali di caratteri, cosa anche oggi impossibile; infatti, i generatori di numeri casuali, creano numeri con proprietà non del tutto casuali.

Figura 2.3: Libriccino contente chiavi usa e getta per il cifrario di Vernam.

2.3 Crittografia simmetrica contemporanea

In questa sezione ci limiteremo a fornire una lista non esaustiva di sistemi a chiave simmetrica relativamente recenti ed ancora oggi utilizzati per le comunicazioni digitali. Non è nostra intenzione entrare nei dettagli, poiché questo richiederebbe troppo spazio. Per chi volesse approfondire, consigliamo [24].

Nell'enorme pletora di crittosistemi a chiave simmetrica, grande predominanza hanno avuto i sistemi a pacchetto, in cui i messaggi vengono codificati in sottopacchetti di lunghezza fissata. Tra questi citiamo i metodi *ECB* (Electronic CodeBook mode), *CFB* (Chiper FeedBack mode) e i crittositemi *affini*.

Un sistema crittografico di notevole importanza è quello di *Feistel*, poiché da esso nacque il più blasonato sistema *DES* (Data Encryption Standard) per cui vale la pena spendere qualche parola.

Il codice DES fu progettato dall'IBM (International Business Machines) nel 1975 e venne introdotto come sistema ufficiale per le comunicazione nel suolo americano. Si tratta di un sistema misto che consta di 4 operazioni fondamentali (permutazione, espansione, sostituzione e permutazione di ciclo) prima di ottenere la codifica del testo. La chiave usata ha una lunghezza di 64 bit di cui 8 di controllo, per un totale di 2^{56} chiavi possibili tra cui scegliere. Quindi, se Eve volesse violare il DES con forza bruta, dovrebbe provare alla peggio 2^{56} chiavi. Sembra un numero gigantesco, e in effetti lo era fino agli anni '90, ma nel 1998, il DES venne violato. Per tale motivo fu aggiornato con il *TripleDES*.

Lo sviluppo del DES era stato tenuto segreto, per cui il NIST (National Institute of Standards and Technology) annunciò un concorso pubblico con lo scopo di determinare un nuovo standard di cifratura simmetrica capace di sbaragliare la concorrenza del DES, e che potesse adattarsi alle innovazioni tecnologiche del nascente millennio. La ricerca si concluse con la scelta del metodo Rijndael, ribattezzato *AES* (Advanced Encryption Standard). Tale sistema utilizza chiavi di 128 bit, sufficienti a garantire una notevole sicurezza.

2.4 Crittografia moderna

Se si vuole comunicare in modo sicuro con qualcuno, utilizzando un sistema crittografico, una buona idea è quella di mantenere segreto il metodo impiegato (e.g. cifrario di Cesare); tuttavia, non sempre è possibile farlo.

Nel 1976, Diffie ed Hellman introdussero il concetto di *crittografia a chiave pubblica* (o *asimmetrica*) [14]. Il lavoro dei due crittografi segnò l'inizio della crittografia moderna suscitando grande interesse nella comunità scientifica. Infatti, il protocollo proposto era concepito in modo tale da non mantenere tutti suoi componenti segreti, anzi la pubblica disponibilità di alcuni parametri è necessaria.

Il funzionamento dei crittositemi a chiave pubblica non è di difficile comprensione. Immaginiamoci una comunità di persone intente a comunicare. Ciascun utente sceglie una funzione crittografica dipendente da alcuni parametri; divulga quei parametri che permettono di codificare i messaggi a lui indirizzati e tiene segreti gli altri. Dunque, Alice può decriptare i messaggi che le arrivano da Bob. Un malintenzionato, capace di intercettare questi messaggi criptati, non potrebbe leggere il testo originale poiché non dispone dei parametri segreti di Alice.

Il doppio lucchetto

Una esemplificazione di un sistema a chiave pubblica è data dal noto indovinello del *paese dei ladri*. Il testo recita così:

> In un paese tutti gli abitanti sono ladri. Non si può camminare per strada con degli oggetti senza che vengano rubati. L'unico modo per spedire qualcosa senza che venga rubato è di metterlo in una cassaforte chiusa con un lucchetto, che è l'unica cosa che in questo paese non viene rubata (mentre le casseforti aperte e i lucchetti vengono rubati). Alla nascita, ogni abitante riceve una cassaforte e un lucchetto di cui possiede l'unica copia della chiave. Ogni cassaforte può essere chiusa con più lucchetti ma la chiave non è cedibile e

non può essere portata fuori casa perché verrebbe rubata. Inoltre, non si può in alcun modo fare una copia delle chiavi. Come può Alice spedire il regalo di compleanno a Bob?

La soluzione è presto detta.

- Alice mette il regalo nella cassaforte che chiude con il suo lucchetto e poi spedisce il tutto a Bob.
- Bob aggiunge il suo lucchetto e rispedisce la cassaforte ad Alice.
- Alice toglie il suo lucchetto e manda la cassaforte a Bob.
- Infine Bob, può togliere il suo lucchetto e scartare il regalo di Alice.

La sicurezza di questo sistema, detto del *doppio lucchetto*, risiede nel fatto che le chiavi per aprire i due lucchetti siano in possesso solo dei rispettivi proprietari. Tuttavia, nelle applicazioni, ci sono troppi scambi della stessa informazione tra i due utenti, rendendo tracciabili eventuali falle nel sistema.

2.4.1 Il problema del Logaritmo Discreto

Uno dei dogmi fondamentali della crittografia moderna è quello di costruire funzioni che siano computazionalmente semplici in un verso, ma di impossibile (o quasi) costruzione nell'altro. Per esempio, se Alice vuole comunicare con Bob, allora sarebbe utile l'uso di una funzione f di cifratura, la quale cifra il messaggio M in un tempo breve (al più polinomiale). Tuttavia, la malintenzionata Eve, potrebbe intercettare il messaggio cifrato $f(M)$ e, per ottenere il testo originale, dovrebbe calcolare $f^{-1}(M)$. Proprio il calcolo di f^{-1} deve essere computazionalmente difficile. Questo è ciò che distingue un crittositema simmetrico da uno asimmetrico.

In tale ambito, grande fortuna hanno avuto sistemi la cui sicurezza si basa sulla presunta difficoltà nel risolvere il logaritmo discreto. Diamo una definizione di questo problema.

Definizione 2.4 (DLP)**.** Sia G un gruppo (moltiplicativo) e sia $g \in G$. Il *DLP* è dato $h \in \langle g \rangle$, trovare il più piccolo k tale che $g^k = h$.

La definizione sopra data è la più generale possibile e può essere adattata al contesto dei campi finiti ($G = \mathbb{F}_q^\times$) o a quello delle curve ellittiche ($G = E(\mathbb{F}_q)$) (per quest'ultima definizione rimandiamo alla sezione 2.5.1).

Definizione 2.5 (DLP in campi finiti)**.** Sia p un primo e sia g un generatore del gruppo $\mathbb{Z}_p^\times$. Il *DLP* è dato $h \in \mathbb{Z}_p^\times$, trovare k tale che $g^k \equiv h \bmod p$

Esistono numerosi algoritmi capaci di risolvere il problema del logaritmo discreto. Per esempio c'è l'algoritmo di *Shanks* (*Baby-Step Giant-Step*), il metodo *ρ di Pollard*, *l'Index calculus* per i gruppi moltiplicativi di campi finiti, e molti altri ancora. Per un descrizione dettagliata di questi algoritmi si possono consultare [24, 49].

Ma se esistono dei modi per risolvere il DLP, come è possibile che i sistemi crittografici fondati su questo problema siano considerati sicuri? In realtà il DLP è sempre risolvibile

teoricamente, ben altra cosa è la pratica. Al giorno d'oggi, non è ancora stato trovato un algoritmo, già implementabile, capace di risolvere il DLP in un tempo accettabile. Infatti, neanche l'unione dei più potenti supercomputer al mondo potrebbe risolvere il problema del logaritmo discreto in "breve" tempo. Ecco perché i crittosistemi che si basano sul DLP sono considerati ad altissima sicurezza.

Per esempio, si stima che la prossima generazione di supercomputer possa toccare i 10^{21} calcoli al secondo [58]. Supponendo di voler risolvere un DLP tramite forza bruta su $\mathbb{Z}_p^\times$ dove $p \approx 2^{128} \approx 10^{39}$, allora dovremmo testare circa 10^{39} numeri. Un calcolo rapido ci dice che un supercomputer impiegherebbe circa 10^{18} secondi per verificare tutti i confronti. Questo numero da solo non dice niente, ma basta confrontarlo con i secondi passati dall'estinzione dei dinosauri (circa 10^{16}) per rendersi conto della portata computazionale del problema del logaritmo discreto.

Tuttavia, va sottolineato che il progresso tecnologico sta portando allo sviluppo dei computer quantistici i cui metodi di calcolo si basano sul *Quantum Turing Machine* (QTM) [3]. Grazie alla potenza di calcolo di questi computer, sarebbe possibile risolvere il DLP utilizzando gli algoritmi proposti da Schor [41, 40] e da Proos e Zalka (per curve ellittiche) [35]. (S)fortunatamente, la realizzazione della QTM non pare essere raggiungibile nel prossimo futuro.

2.4.2 Lo scambio di chiavi di Diffie-Hellman

La forza e l'importanza dei crittosistemi a chiave pubblica risiede nella possibilità di poter scambiare, in modo sicuro, chiavi da utilizzare in un sistema tradizionale. Infatti, se Alice e Bob vogliono comunicare utilizzando il cifrario di Cesare, allora devono concordare *a priori* la chiave (lettera/numero) da usare. Supponiamo che Alice e Bob non possano vedersi fisicamente e che l'utilizzo di un corriere sia troppo dispendioso in termini di tempo. L'unica possibilità è la comunicazione tramite canali pubblici (insicuri). Ma come fanno a essere certi che nessuno intercetti il messaggio con su scritta le chiave? Con l'omonimo algoritmo, Diffie ed Hellman proposero una risposta a questa domanda [14].

Il protocollo si divide in quattro parti: la generazione dei parametri del sistema; la generazione delle chiavi; lo scambio e il calcolo della chiave segreta.

- *Parametri del sistema:* Alice e Bob concordano (pubblicamente) su un numero primo grande p ed un generatore $g \in \mathbb{Z}_p^\times$.

- *Generazione delle chiavi:* Alice sceglie in modo casuale un elemento $a \in \{2, \ldots, p-2\}$ e calcola

  $$k_A \equiv g^a \bmod p$$

  Ugualmente, Bob sceglie in modo casuale un elemento $b \in \{2, \ldots, p-2\}$ e calcola

  $$k_B \equiv g^b \bmod p$$

  Il valore a (rispettivamente b) è la *chiave privata* di Alice (rispettivamente Bob), mentre k_A (rispettivamente k_B) è la *chiave pubblica* di Alice (rispettivamente Bob).

- *Scambio:* Alice e Bob si trasmettono a vicenda le chiavi pubbliche.

- *Chiave segreta:* Alice calcola $K_A \equiv g^{ab} \equiv (k_B)^a \bmod p$. Bob calcola $K_B \equiv g^{ab} \equiv (k_A)^b \bmod p$. Quindi, $K = K_A = K_B$ è la *chiave segreta*.

Per le proprietà dell'aritmetica modulare, $(k_B)^a \bmod p$ e $(k_A)^b \bmod p$ coincidono. Quindi, Alice e Bob condividono la stessa chiave K che può essere utilizzata in cifrari simmetrici quali il DES o AES.

Uno schema figurativo dello scambio di chiavi è rappresentato in figura 2.4, mentre ora presentiamo una esemplificazione, con numeri piccoli, del protocollo.

Esempio 4. Scambio di chiavi di Diffie-Hellman.

- Alice e Bob scelgono e pubblicano $p = 151$ e $g = 6$.
- Alice sceglie come chiave privata $a = 123$ e calcola $k_A \equiv g^a \equiv 83 \bmod 151$. Quindi invia a Bob 83.
- Bob sceglie come chiave privata $b = 8$ e calcola $k_B \equiv g^b \equiv 43 \bmod 151$. Quindi invia ad Alice 43.
- Alice calcola $K_A \equiv (g^b)^a \equiv 44 \bmod 151$.
- Bob calcola $K_B \equiv (g^a)^b \equiv 44 \bmod 151$.

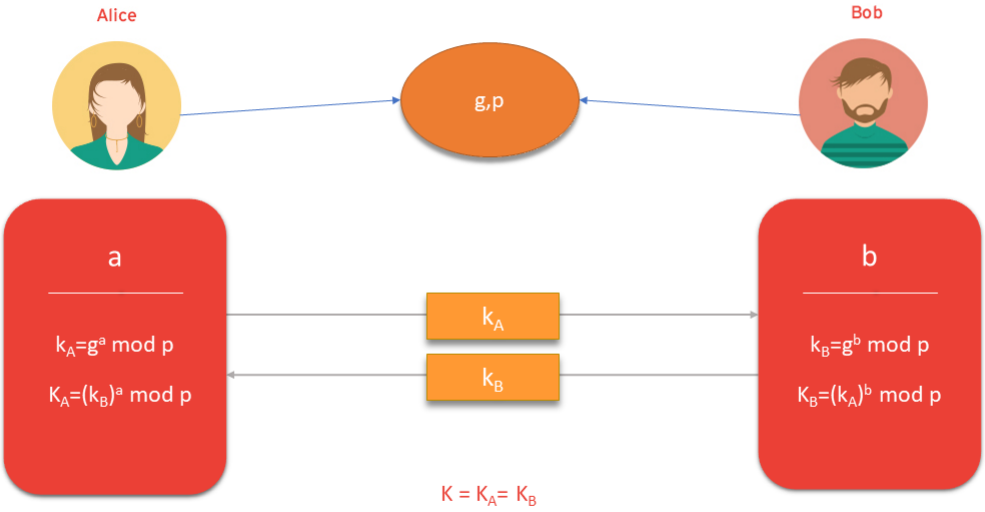


Figura 2.4: Schema del protocollo di Diffie-Hellman

Sicurezza

La nostra Eve, che vuole scoprire quale è la chiave segreta (g^{ab}) dovrebbe risolvere il seguente problema.

Definizione 2.6. Sia p un primo e g un generatore del gruppo $\mathbb{Z}_p^\times$. Il *Problema di Diffie-Hellman (DHP)* è, dati gli elementi $g^a, g^b \in \mathbb{Z}_p^\times$, calcolare g^{ab}.

La sicurezza del protocollo risiede nella presunta difficoltà nel risolvere il problema del logaritmo discreto (Sezione 2.4.1). Infatti, chiunque sia in grado di risolvere tale problema allora può risolvere il DHP. Tuttavia, è congetturabile (ma non dimostrato) che i due problemi siano equivalenti [10], [9].

Sfortunatamente, l'algoritmo di Diffie-Hellman è vulnerabile ad attacchi "Man in the middle". Supponiamo che Alice mandi la sua chiave pubblica a Bob. Eve intercetta tale chiave e spedisce a Bob un falso, ovvero una chiave pubblica creata da Eve tramite la chiave privata di quest'ultima. A questo punto Eve può intercettare i messaggi scritti da Bob, li decifra (e li altera se vuole) e poi li ri-cifra con la chiave pubblica di Bob. Infine, spedisce i messaggi ad Alice.

2.4.3 I sistemi di ElGamal

Altri protocolli la cui sicurezza si basa sul DLP sono i sistemi concepiti da Tahar ElGamal nel 1985 [17]. In particolare, il crittografo egiziano progettò un sistema per la cifratura e uno schema per la firma digitale.

Cifratura di ElGamal

Tale crittosistema consta di quattro momenti: la scelta dei parametri del sistema; la generazione delle chiavi; cifratura e decifratura del messaggio.

- *Parametri del sistema:* Bob sceglie (pubblicamente) un numero primo grande p e un generatore g del gruppo ciclico $\mathbb{Z}_p^\times$.

  Generazione delle chiavi: Bob sceglie la chiave privata $b \in \{2, \dots, p-1\}$; calcola $k_B \equiv g^a \bmod p$; pubblica la tripla (p, g, k_B) (chiave pubblica).

- *Cifratura:* Alice vuole mandare a Bob il messaggio M, dove M è la rappresentazione tramite un numero intero minore di p del messaggio in lettere (oppure si può applicare una funzione di hash al messaggio A). Prima di tutto, Alice sceglie a caso un numero $k \in \{2, \dots p-1\}$. Dopodiché, calcola

  $$r \equiv g^k \bmod p \quad \text{e} \quad s \equiv M k_B^k \bmod p$$

  Quindi invia a Bob la coppia (r, s).

- *Decifratura:* Bob riceve (r, s) e decripta il messaggio calcolando

  $$M \equiv \frac{s}{r^b} \bmod p$$

Il procedimento appenda descritto è corretto. Infatti,

$$\frac{s}{r^b} \equiv \frac{Mk_B^k}{(g^k)^b} \equiv \frac{M(g^b)^k}{g^{kb}} \equiv \frac{Mg^{kb}}{g^{kb}} \equiv M \bmod p$$

Uno schema figurativo è apprezzabile in figura 2.5, mentre un esempio di esecuzione è proposto qui sotto.

Esempio 5. Schema di cifratura di ElGamal.

- Bob sceglie $p = 2357$, $g = 2$ e come chiave privata $b = 1751$. Calcola $k_B \equiv g^b \bmod p$ e condivide la chiave pubblica

$$(p, g, k_B) = (2357, 2, 1185 \bmod 2357)$$

- Alice vuole mandare il messaggio "CIAO" a Bob. Codifica il testo come $M = 2035$. Quindi, sceglie a caso l'intero k. Supponiamo $k = 1520$. Alice calcola

$$r \equiv g^k \equiv 1430 \bmod 2357 \quad \text{e} \quad s \equiv Mk_B^k \equiv 697 \bmod 2357$$

  Infine, manda a Bob la coppia $(1430, 697)$.

- Bob riceve $(1430, 697)$ e calcola

$$M \equiv \frac{s}{r^b} \equiv \frac{697}{1430^{1751}} \equiv 2035 \bmod 2357$$

Sicurezza

La nostra malintenzionata Eve conosce p, g, k_A, k_B e può intercettare la coppia (r, s). Per decriptare il messaggio, ha bisogno o della chiave privata di Bob b oppure del numero k. Infatti, conoscendo quest'ultimo valore, Eve può calcolarsi M da $s \equiv MK_B^k \bmod p$. Per trovare b, Eve deve risolvere il problema del logaritmo discreto, poiché $k_B \equiv g^b \bmod p$. Nel secondo caso, si trova ancora a dover fronteggiare lo stesso problema, dato che $r \equiv g^k \bmod p$.

Notiamo che se si dovesse mandare più di un messaggio è buona cosa cambiare il parametro segreto k. Infatti, supponendo che Eve riesca ad intercettare due messaggi (r, s_1) e (r, s_2) generati con lo stesso k, potrebbe calcolare

$$\frac{s_1}{s_2} \equiv \frac{M_1 k_B^k}{M_2 k_B^k} \equiv \frac{M_1}{M_2} \bmod p$$

Quindi, sarebbe in grado di risalire a M_2 conoscendo M_1 (o viceversa).

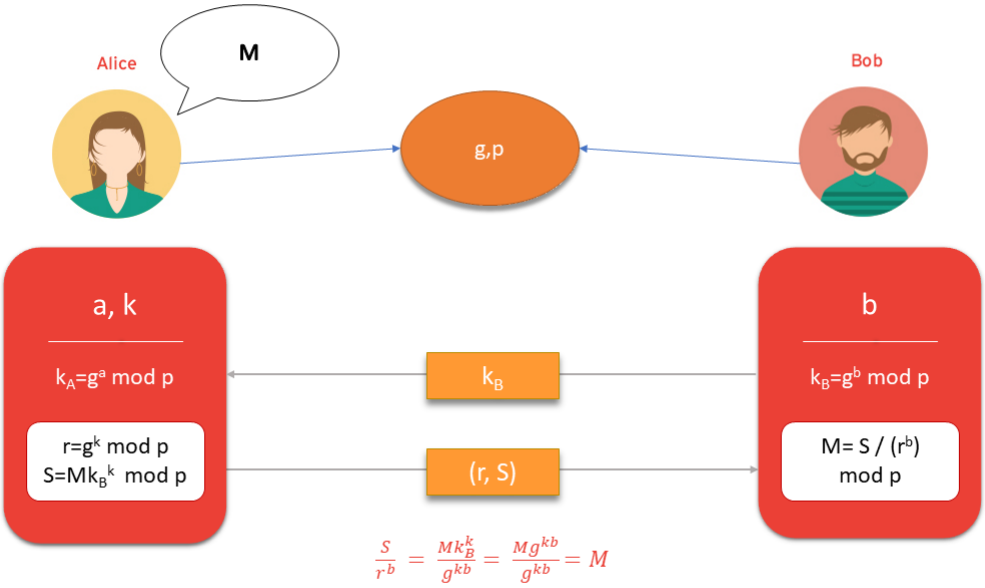


Figura 2.5: Schema di cifratura di ElGamal

Firma digitale di ElGamal

Parlando della sicurezza del protocollo Diffie-Hellman, abbiamo detto che esso è vulnerabile ad un attaccato "Man in the middle". Questo accade perché non è possibile autenticare la provenienza dei messaggi. Infatti, Alice compie un atto di fede nel decifrare i messaggi provenienti da Bob, perché, di fatto, non ha la certezza che siano stati mandati da egli stesso.

Il problema di certificare la propria identità nel mondo virtuale è alla base del concetto di firma digitale. Tale concetto è analogo alla firma posta in calce ad un documento cartaceo. La nostra scrittura funge da impronta digitale ed assicura la provenienza nonché l'autenticità del documento stesso. La crittografia a chiave pubblica ha reso possibile traslare questa idea "terrena" nel mondo della comunicazione digitale.

Un esempio di metodo di firma è basato sul crittosistema di ElGamal presentato poc'anzi. Anche questo schema si basa su quattro parti: scelta dei parametri di sistema; generazione delle chiavi; firma; verifica della firma.

- *Parametri del sistema:* Alice sceglie (pubblicamente) un numero primo grande p e un generatore g del gruppo ciclico $\mathbb{Z}_p^\times$.
- *Generazione delle chiavi:* Alice sceglie la chiave privata $a \in \{2, \dots, p-2\}$; pubblica il valore (p, g, k_A) (chiave pubblica) dove $k_A \equiv g^a \bmod p$.
- *Firma:* Alice vuole firmare un messaggio M (codificato tramite un numero intero minore di p). Prima di tutto, sceglie casualmente un intero $k \in \{2, \dots, p-2\}$ tale che $\gcd(k, p-1) = 1$. Quindi, calcola

$$r \equiv g^k \bmod p$$

A questo punto, Alice risolve in s la congruenza

$$M \equiv ar + ks \bmod p - 1$$

(la congruenza è risolvibile tramite l'algoritmo di Euclide, poiché k è invertibile modulo $p-1$ per ipotesi). Dunque

$$s \equiv k^{-1}(M - ar) \bmod p - 1$$

Infine, Alice manda la tripla (M, r, s) a Bob, dove (r, s) è la firma.

- *Verifica:* Bob riceve (M, r, s) e per verificare l'identità del mittente calcola

$$u_1 \equiv (g^a)^r r^s \bmod p \quad \text{e } u_2 \equiv g^M \bmod p$$

La firma è accettata soltanto se $u_1 = u_2$.

Vale la pena notare che la correttezza della verifica si basa sul Piccolo Teorema di Fermat.

Teorema 5. *Sia p un numero primo e sia a un numero intero, allora*

$$a^p \equiv a \bmod p$$

In particolare, se p non divide a, allora

$$a^{p-1} \equiv 1 \bmod p$$

Dunque, per ipotesi $\gcd(g, p) = 1$, quindi p non divide g. Allora $g^{p-1} \equiv 1 \bmod p$, inoltre $M - (ar + kv) = (p-1)h$ per un certo $h \in \mathbb{Z}$. Mettendo insieme le cose otteniamo che

$$g^M \equiv g^{M-(ar+kv)} g^{ar+kv} \equiv g^{(p-1)h} g^{ar+kv} \overset{Fermat}{\equiv} g^{ar+kv} \equiv (g^a)^r r^v \bmod p$$

Vediamo un esempio di firma digitale. Un rappresentazione schematica è visibile nella figura 2.6.

Esempio 6. Schema di firma digitale di ElGamal.

- Gli utenti scelgono e pubblicano $p = 2357$, $g = 2$.
- Alice, un utente del sistema, sceglie come chiave privata $a = 1751$. Calcola e condivide la chiave pubblica

$$k_A \equiv g^a \equiv 1185 \bmod 2357$$

- Alice vuole mandare il messaggio "CIAO" a Bob. Codifica il testo come $M = 2035$. Quindi, sceglie a caso un intero k tale che $\gcd(k, p-1) = 1$. Supponiamo $k = 1521$. Alice calcola

$$r \equiv g^k \equiv 503 \bmod 2357$$

Con l'algoritmo di Euclide, Alice scopre che

$$\gcd(1521, 2356) = 1 = 1521 \cdot (-759) + 2356 \cdot 490$$

da cui

$$s \equiv k^{-1}(M - ar) \equiv 1597(2035 - 1751 \cdot 503) \equiv 1058 \bmod 2356$$

Infine, manda a Bob $(2035, 503, 1058)$.

- Bob riceve $(2035, 503, 1058)$ e calcola

$$(g^a)^r r^s \equiv 1230 \bmod 2357$$

che coincide con

$$g^M \equiv 1230 \bmod 2357$$

Quindi Bob conclude che il messaggio viene da Alice.

Sicurezza

Poiché nella firma viene usata la chiave segreta a di Alice ciò prova l'identità del mittente. Infatti, ogni altro utente, per risolvere la congruenza $M \equiv ar + ks \bmod p - 1$, dovrebbe saper determinare a a partire da g^a e p. Ovvero, dovrebbe risolvere un DLP . Chiaramente, anche il parametro k deve restar segreto, in caso contrario, Eve potrebbe risalire alla chiave privata di Alice a semplicemente intercettando il messaggio e la firma. Tuttavia, determinare k è arduo poiché esso richiede la risoluzione di un logaritmo discreto

Come per lo schema di cifratura, in caso di firme multiple è consigliato generare due diversi k. Infatti, collezionando (r, s_1) ed (r, s_2) nati con lo stesso k, Eve può calcolare

$$(s_1 - s_2) \cdot k \equiv M_1 - M_2 \bmod p - 1$$

e da qui Eve potrebbe risalire alla chiave privata a da s_1 o s_2. Usiamo la parola "potrebbe" poiché non è immediato ricavarsi la chiave di Alice dalla congruenza sopra; infatti, i calcoli sono fatto modulo un numero pari (non primo) e ciò porta con se degli ovvi problemi di invertibilità. In particolare, non è detto che $s_1 - s_2$ sia invertibile modulo $p - 1$.

A discapito della descrizione originale dell'algoritmo, nelle applicazioni è preferibile applicare una funzione di hash h al messaggio M. Infatti, senza di essa, sarebbe facile per Eve generare un messaggio falsificato M' tale che (M', r, s) sia un firma accettata da Bob (supponendo che Eve abbia intercettato (M, r, s) da Alice). Con una funzione di

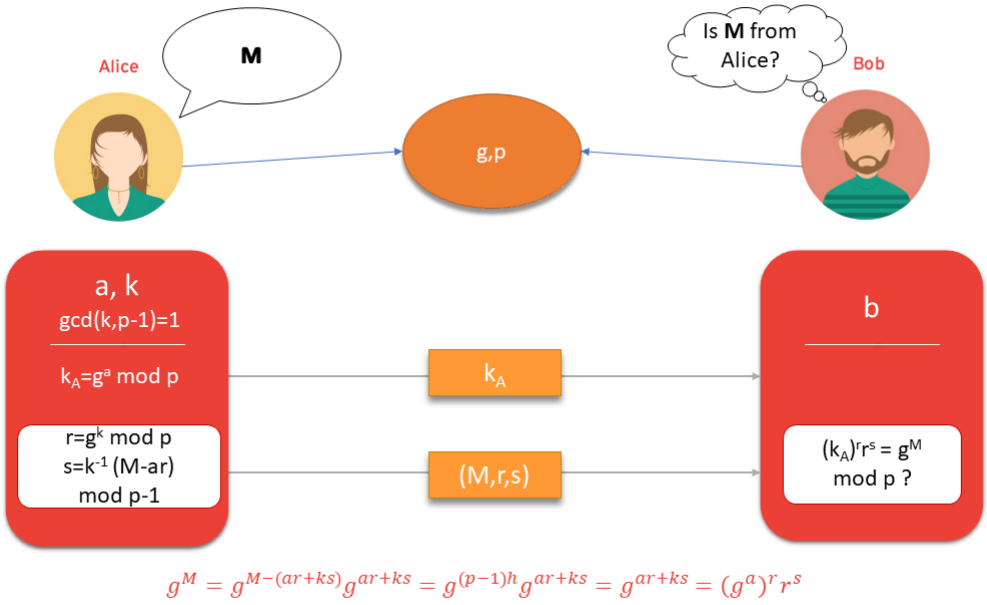


Figura 2.6: Schema di firma digitale di ElGamal

hash, tuttavia, Alice non firma direttamente M ma $h(M)$. Quindi Eve, per farsi accettare da Bob, dovrebbe trovare un M' tale che $h(M) = h(M')$, ma le funzioni di hash sono resistenti alle collisioni (si veda l'appendice A per maggiori dettagli). Questo attacco va sotto il nome di *falsificazione esistenziale*.

2.4.4 Digital Signature Algorithm

Nell'ambito dei protocolli di firma digitale, il *Digital Signature Algorithm* (DSA) è quello probabilmente più utilizzato al livello mondiale insieme alla sua variante ellittica. Esso fu proposto dal NIST nel 1991 sotto richiesta del governo americano. Tale schema, ricorda in gran parte quello di ElGamal pur avendo uno scheletro profondamente diverso.

Il protocollo presenta le solite quattro parti: scelta dei parametri; generazione delle chiavi, firma e verifica.

- *Parametri del sistema:* Alice sceglie un primo p, tale che $2^{L-1} < p < 2^L$ dove L è la lunghezza in bit di p. Dopodiché, sceglie un divisore primo q di $p-1$ tale che $2^{N-1} < q < 2^N$ dove N è la lunghezza in bit di q. Infine, sceglie un generatore g del gruppo $\mathbb{Z}_q^\times$ modulo p, ovvero $g^q \equiv 1 \bmod p$.

  Il NIST consiglia dei valori sicuri per L ed N in [31]. Per completezza, riportiamo tali valori nella tabella 2.2.

- *Generazione delle chiavi:* Alice sceglie casualmente un intero $a \in \{2, \ldots, q-1\}$. Quindi calcola $k_A \equiv g^a \bmod p$. La chiave pubblica è (p, q, g, k_A).

- *Firma* Alice vuole firmare un messaggio M. Per farlo sceglie un numero casuale $k \in \{2, \ldots, q-1\}$ e calcola

$$r \equiv (g^k \bmod p) \bmod q \quad \text{e} \quad s \equiv k^{-1}(h(M) + ar) \bmod q$$

dove h è una funzione di hash che restituisce N bit del messaggio. Se $r = 0$ oppure $s = 0$, allora il processo di firma va ripetuto.

Quindi, invia a Bob la tripla (M, r, s) dove (r, s) costituisce la firma digitale.

- *Verifica:* Bob riceve (M, r, s) e calcola

$$u_1 \equiv s^{-1} \cdot h(M) \bmod q \quad \text{e} \quad u_2 \equiv s^{-1} \cdot r \bmod q$$

Infine, calcola

$$v \equiv (g^{u_1} \cdot k_A^{u_2} \bmod p) \bmod q$$

ed accetta la firma soltanto se $v = r$.

Se la firma è stata generata correttamente, allora, da $s \equiv k^{-1}(h(M) + ar) \bmod q$, otteniamo

$$\begin{aligned} h(M) &\equiv -ar + k \bmod q \\ s^{-1}h(M) &\equiv -ars^{-1} + k \bmod q \end{aligned}$$

da cui

$$\begin{aligned} k &\equiv s^{-1}h(M) + ars^{-1} \\ &\equiv u_1 + au_2 \bmod q \end{aligned}$$

Dunque

$$r = (g^k \bmod p) \bmod q = (g^{u_1+au_2} \bmod p) \bmod q = (g^{u_1}k_A^{u_2} \bmod p) \bmod q = v$$

Osservazione 2. A differenza dello schema di ElGamal, i calcoli sono fatti modulo q che è piccolo a parità di p ($N < L$). Questo rende il DSA più veloce rispetto al suo predecessore.

Vediamo un esempio con piccoli numeri. Come sempre è disponibile uno schema figurativo dell'algoritmo nella figura 2.7.

Esempio 7. Schema di firma DSA.

- Alice sceglie $p = 59$ e $q = 29$. Quindi, come generatore sceglie $g = 3$ notando che $3^{29} \equiv 1 \bmod 59$.
- Ora, Alice sceglie $a = 7$ e calcola $k_A = 3^7 \equiv 4 \bmod 59$. Quindi pubblica (p, q, g, k_A).
- Alice vuole firmare il messaggio "CIAO", la cui impronta è $h(M) = 26$. Quindi sceglie $k = 10$ e calcola

$$r = (3^{10} \bmod 59) \equiv 20 \bmod 29 \quad \text{e} \quad s = 3 \cdot (26 + 7 \cdot 20) \equiv 5 \bmod 29$$

Infine, invia a Bob (M, r, s).

- Bob, una volta ricevuta la tripla, calcola $s^{-1} \equiv 6 \bmod 29$ da cui

$$u_1 = 6 \cdot 26 \equiv 11 \bmod 29$$

e

$$u_2 = 6 \cdot 20 \equiv 4 \bmod 29$$

Infine calcola

$$v = (3^{11} \cdot 4^4 \bmod 59) \equiv 20 \bmod 29$$

e accetta la firma poiché $v \equiv r \bmod 29$.

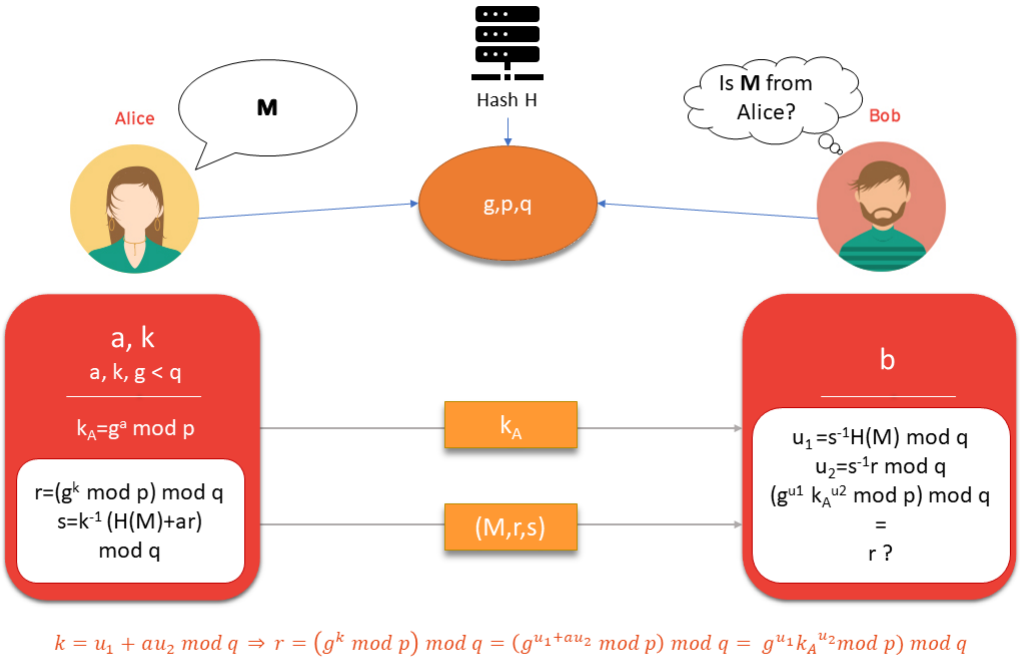


Figura 2.7: Schema DSA

Sicurezza

Come per ElGamal, se Eve possiede la chiave privata a può firmare spacciandosi per Alice. Tuttavia, per risolvere la congruenza $h(M) \equiv -ar + k \bmod q$, Eve dovrebbe saper calcolare a a partire da k_A e p . Ma questo è un DLP. Se inoltre Eve riesce ad intercettare (r, s) allora per calcolare a avrebbe bisogno di k, ma quest'ultimo è calcolabile ancora mediante logaritmo discreto. Notiamo inoltre la presenza di una funzione di hash h, che impedisce eventuali privazioni di identità.

Infine, bisogna ricordarsi di utilizzare k diversi per firme diverse, altrimenti, si potrebbe risalire a k conoscendo s_1 ed s_2 esattamente come detto per ElGamal. Stavolta

L	N
1024	160
2048	224
2048	256
3072	256

Tabella 2.2: Valori consigliati dal NIST per la coppia (L, N)

sarà più semplice per Eve risalire alla chiave privata di Alice collezionando due firme consecutive, poiché i calcoli sono fatti modluo q che è un numero primo.

DSA come crittosistema completo

IL DSA è un algoritmo strettamente legato alla sola firma digitale. Tuttavia, è possibile sfruttarlo congiuntamente a un cifrario simmetrico per costruire un sistema crittografico onnicomprensivo, cioè completo di generazione e scambio della chiavi, cifratura/decifratura e firma dei messaggi. Tale costruzione fu proposta da Young e Yung in [57]. Vediamo, quindi, come fabbricare tale sistema.

Ricordiamo che nel DSA si sceglie a caso k e si calcola $r \equiv (g^k \bmod p) \bmod q$. Conoscendo (r, s) si può risalire a $g^k \bmod p$. Questo permettere di implementare un scambio di chiavi alla Diffie-Hellman nel seguente modo. Detta y_B la chiave pubblica di Bob, Alice calcola $z \equiv y_B^k \bmod p$ dove k è casuale. A questo punto, l'elemento z può essere usato come chiave per cifrare un messaggio M usando, per esempio, AES. Ora, Alice firma il testo cifrato $AES(M)$ usando il protocollo DSA con chiave privata $x_A = k$, ottenendo (r, s). Con questa coppia, Bob può recuperare $g^k \bmod p$, ma allora, basta che usi la sua chiave privata x_B per ottenere z; infatti $z \equiv y_B^k \equiv (g^{x_B})^k \equiv (g^k)^{x_B}$. Infine, Bob può decifrare il messaggio $AES(M)$ con z ed ottenere M.

2.4.5 RSA

Il re dei crittositemi di cifratura a chiave pubblica è probabilmente il codice *RSA*, dalle iniziali di Rivest, Shamir e Adlmean che lo progettarono nel 1978 [37]. Riportiamo per completezza il funzionamento di tale metodo, senza perderci in dettagli, poiché RSA non è essenziale per il proseguito di questa tesi.

Supponiamo che Bob voglia scambiare un messaggio M con Alice. La cifratura RSA funziona come di seguito spiegato.

- Alice sceglie due numeri primi distinti p e q abbastanza grandi da garantire la sicurezza del metodo. Dopodiché, calcola $n = pq$.

- Alice, calcola $\phi(n) = (p-1)(q-1)$, dove ϕ è la funzione di Eulero. Quindi sceglie un numero $e \in \mathbb{N}$ coprimo con $\phi(n)$. La coppia (n, e) sarà la chiave pubblica di Alice.

- La ragazza calcola $d \in \mathbb{Z}^{\times}_{\phi(n)}$, il quale è l'inverso modulo $\phi(n)$ di e. La tripla (p, q, e) deve essere tenuta segreta e costituisce la chiave privata.

- A questo punto Bob dispone della chiave pubblica di Alice e codifica il messaggio calcolando

$$c \equiv M^e \bmod n$$

che invia alla ragazza.

- Quest'ultima, per codificare il messaggio utilizza la funzione inversa determinata dal parametro d. Quindi

$$M \equiv c^d \bmod n$$

Si verifica velocemente che

$$c^d \equiv (M^e)^d \equiv M^{ed} \equiv M \bmod n$$

dove l'ultima relazione vale per il Piccolo teorema di Fermat e per il teorema cinese del resto.

Concludiamo facendo notare che, a differenza dei precedenti sistemi, la sicurezza di RSA si basa sulla presunta difficoltà di scomporre n nei suoi fattori primi.

2.5 Crittosistemi su Curve Ellittiche

Come già accennato nel capitolo 1, la teoria delle curve ellittiche non è giovane, ma risale addirittura al Seicento; tuttavia, solo nell'ultimo mezzo secolo si è intuita una sua potenziale applicazione in ambito crittografico.

La comparsa di questi oggetti matematici sul panorama della sicurezza digitale si deve ai lavori di Koblitz [20] e Miller [26] del 1985. Da quel momento, lo studio delle curve ellittiche conobbe un incremento importante, tanto è vero che, ad oggi, sono ampiamente diffuse. Per esempio, WhatsApp e Telegram basano la loro sicurezza su un curva chiamata Curve25519, sfruttata per lo scambio di chiavi tra gli utenti. La firma digitale usata per i Bitcoin è costruita sulla curva secp256k1. Anche il TLS (una tecnologia che garantisce la sicurezza ad una connessione ad internet e protegge dati sensibili), presente in Chrome, è fondato su sistemi ellittici. Tuttavia, non si fraintenda che le curve ellittiche detengano il monopolio dei sistemi crittografici. Protocolli come RSA e AES sono sono ancora utilizzati, per esempio, per cifrare e decifrare i messaggi. Il motivo è che questi sono più rapidi nello scambio di informazioni, fornendo comunque una adeguata sicurezza. Ad esempio, si pensi ad una partita di calcio trasmessa in diretta da una mittente televisiva a pagamento, se le immagini arrivassero dopo alcuni secondi si perderebbe la percezione della visione in live; tuttavia si vorrebbe mantenere criptato il segnale in modo da proibire divulgazioni non autorizzate.

Da un punto di vista teorico, le curve ellittiche sono uno strumento più flessibile in cui costruire protocolli crittografici. Infatti, con un minor dispendio di memoria (in termini di bit) si possono ottenere crittosistemi di pari sicurezza rispetto a quelli convenzionali (RSA, DSA) [48]. Tuttavia, non tutte le curve ellittiche sono utilizzabili per scopi crittografici. Per esempio, è stato dimostrato che, se $|E(\mathbb{F}_q)| = q$, allora l'analogo ellittico del DLP è velocemente risolvibile [7]. A tal proposito il NIST (National Institute of Standards and

Technology) e Certicom (azienda specializzata in sicurezza informatica) hanno pubblicato una lista di curve considerate sicure [11, 31, 36].

Infine, i vantaggi delle curve ellittiche non sono solo computazionali. Per esempio, il metodo Index calculus usato per risolvere il problema del logaritmo discreto è inappliabile al contesto ellittico.

2.5.1 Un nuovo problema per il Logaritmo Discreto

Nel mondo delle curve ellittiche, il problema del logaritmo discreto pare essere un problema di superiore difficoltà rispetto alla versione su campi finiti. Va detto che non basta lavorare su campi $\mathbb{F}_q$ per q grande per garantire una buona sicurezza, ma bisogna scegliere adeguatamente la curva, affinché essa non presenti anomalie che rendo risolvibile il problema del logaritmo discreto in tempi ragionevoli. Un esempio è fornito dalle curve *supersingolari*, una cui trattazione minuziosa è fornita in [49].

Senza ulteriori indugi, definiamo il problema del logaritmo discreto su curve ellittiche (ECDLP).

Definizione 2.7. Sia $q = p^n$ con p un numero primo ed $n \geq 1$. Sia E una curva ellittica su $\mathbb{F}_q$. L'*ECDLP* è dati $P, Q \in E(\mathbb{F}_q)$, trovare k tale $Q = [k]P$.

Data la somiglianza con la definizione originale, non deve stupire che gli attacchi già presentati nella sezione 2.4.1 valgono anche per l'ECDLP, con i dovuti aggiustamenti (tranne l'Index calculus). Tuttavia, sono stati descritti attacchi specifici per risolvere l'ECDLP che basano la loro forza sul ridurre il problema ellittico ad un normale DLP, definito su un gruppo moltiplicativo di un campo finito. Quindi, citiamo l'attacco *Frey-Rück* e l'attacco *MOV* [49].

2.5.2 Elliptic Curve Diffie-Hellman Keys Exchange

La descrizione dello scambio di chaivi di Diffie-Hellman è del tutto analoga a quella già presentata in 2.4.2, con lievi modifiche per adattarla al contesto ellittico. Vediamo come.

- *Parametri del Sistema:* Alice e Bob concordano (pubblicamente) su una curva ellittica E definita su un campo finito $\mathbb{F}_q$ tale che l'ECDLP sia computazionalmente difficile (per esempio le curve suggerite dal NIST). Inoltre, scelgono un punto $G \in E(\mathbb{F}_q)$, detto *punto iniziale*, tale che l'ordine n del sottogruppo generato da G, sia un primo grande.

- *Generazione delle chiavi:* Alice sceglie in modo casuale un elemento $a \in \{2, \ldots, n-1\}$ (la sua chiave segreta) e calcola il valore $A = [a]G$. Lo stesso compie Bob. Sceglie in modo casuale un elemento $b \in \{2, \ldots, n-1\}$ e calcola il valore $B = [b]G$.

- *Scambio:* Alice e Bob si scambiano rispettivamente A e B.

- *Chiave segreta:* Alice calcola $K_A = [a]B$. Bob calcola $K_B = [b]A$. Ma nelle curve ellittiche vige la proprietà associativa, per cui $K = K_A = K_B$ è la comune chiave segreta.

Osservazione 3. Durante lo scambio delle chiavi A e B, non c'è nessuna necessità di trasmettere entrambe le coordinate del punto. In effetti, conoscendo la coordinata x, si può determinare l'ordinata a meno del segno (per via della forma di Weierstrass (1.2)). Quindi, Alice e Bob possono mandare l'ascissa del punto più un bit che indichi quale segno ha la coordinata y. I vantaggi sono solari: la trasmissione avviene inviando metà dell'informazione, senza perdere il contenuto originario. Quindi, si ottiene una maggiore efficienza senza lasciare pezzi per strada.

Nelle applicazioni, questo è quello che avviene. Ad esempio, in un ECC da 160 bit, si applica una funzione di hash alla coordinata x di K che produce un output di 160 bit. Di questi, 128 bit possono essere utilizzati come chiave per l'AES.

Proponiamo un esempio con piccoli numeri del protocollo ECDH.

Esempio 8. Scambio di chiavi ECDH.

- Alice e Bob scelgono la curva $E : y^2 = x^3 + 2x + 2$ definita sul campo $\mathbb{F}_{17}$. Tale curva forma un gruppo ciclico di ordine $\#E = 19$. Il punto iniziale scelto è $G = (5, 1)$ il cui ordine coincide con quello del gruppo.
- Alice sceglie casualmente $a = 3$ e calcola $A = [3]G = (10, 6)$. Analogamente, Bob sceglie $b = 10$ e calcola $B = [10]G = (7, 11)$
- Alice e Bob si scambiano A e B oppure le sole ascisse 10 e 7 più un bit identificativo.
- Alice calcola $K_A = [3]B = (13, 10)$ mentre Bob calcola $K_B = [10]A = (13, 10)$. A questo punto resta individuata la chiave segreta $K = (13, 10)$.

Sicurezza

Come nella descrizione originale del DH, la sicurezza risiede nel fatto che Eve dovrebbe risolvere un DHP in ambito ellittico (ECDHP).

Definizione 2.8. Sia $q = p^n$ con p un numero primo ed $n \geq 1$. Sia E una curva ellittica su $\mathbb{F}_q$. Sia $G \in E(\mathbb{F}_q)$ il punto iniziale. Il ECDHP è un problema che, dati $[a]G, [b]G$, chiede di calcolare $[ab]G$.

Tale problema, ha una complessità computazionale simile a quella del ECDLP, pertanto difficile da risolvere.

Bisogna sottolineare che, nonostante il DLP e ECDLP abbiamo una radice comune, il logaritmo discreto su curve ellittiche è molto più difficile da violare. In effetti, l'algoritmo più potente contro il DLP, cioè l'Index calculus, non è nemmeno applicabile nel contesto ellittico. Tuttavia, l'ECDH è ancora vulnerabile ad attacchi Man-in-the-middle.

La scelta di un punto G di ordine un numero primo è dovuto al fatto che, altrimenti, l'algoritmo di Pholing ed Hellman per risolvere il logaritmo discreto, romperebbe con relativa velocità l'ECDLP [43].

Infine, notiamo che se le curva non è scelta adeguatamente, esiste un metodo, basato sul *Weil paring*, in grado di ridurre la difficoltà del ECDLP [49].

2.5.3 Elliptic Curve ElGamal Encryption

Presentiamo ora lo schema di cifratura ideato da ElGamal ma adattato al contesto delle curve ellittiche (ECEE).

- *Parametri del sistema:* Bob sceglie una curva ellittica E definita su un campo $\mathbb{F}_q$ in cui il problema dell' ECDLP sia computazionalmente difficile. Inoltre, sceglie un punto $G \in E(\mathbb{F}_q)$ di ordine un primo n grande.
- Generazione delle chiavi: Bob sceglie casualmente un intero $b \in \{2, \dots, n-1\}$ e calcola $B = [b]G$. La chiave pubblica è (E, q, G, B).
- *Cifratura:* Alice vuole mandare il messaggio M a Bob, dove M è la rappresentazione tramite un punto di $E(\mathbb{F}_q)$ del messaggio in lettere. Prima di tutto, Alice sceglie a caso un intero $k \in \{2, \dots, n-1\}$ e calcola

  $$R = [k]G \quad \text{e} \quad S = M + [k]B$$

  Quindi invia a Bob la coppia (R, S).
- *Decifratura:* Bob decripta il messaggio calcolando

  $$M = S - [b]R$$

Il procedimento funziona sempre grazie all'associatività. Infatti,

$$S - [b]R = (M + [k]B) - [b]([k]G) = M + [k]([b]G) - [bk]G = M$$

Osservazione 4. Nell'algoritmo appena presentato abbiamo detto che il messaggio M è codificato mediante un punto di E; tuttavia non è semplice compiere questa operazione affinché essa sia, in un certo senso, invertibile. Ovvero, detta $T : \mathcal{M} \to E(\mathbb{F}_q)$ tale che $T(M) = P$, chi ci assicura che $T^{-1}(M) = P$? Il problema è che le coordinate x di tutti i punti di E, solitamente non sono sufficienti a coprire tutto il campo $\mathbb{F}_q$, creando delle ambiguità quando si cerca di ritrovare il messaggio originario M conoscendo P.

Per ovviare a questo problema, si può modificare l'algoritmo decodificando M mediante un intero minore di n e sostituendo S con

$$s \equiv M \cdot ([k]B)_x$$

dove $([k]B)_x$ denota l'ascissa del punto $[k]B$.

Vediamo un esempio dell'algoritmo con piccoli numeri.

Esempio 9. Schema di cifratura ECEE.

- Bob sceglie la curva $E : y^2 = x^3 + 2x + 2$ definita sul campo $\mathbb{F}_{17}$. Tale curva forma un gruppo ciclico di ordine $\#E = 19$. Il punto iniziale scelto è $G = (5, 1)$ il cui ordine coincide con quello del gruppo.
- Bob sceglie $b = 3$ e calcola $B = [3]G = (10, 6)$. Quindi pubblica (E, q, G, B).
- Alice vuole mandare il messaggio "CIAO" a Bob. Tale messaggio viene prima codificato in un punto di $E(\mathbb{F}_q)$, per esempio "CIAO" $= (9, 16)$. Quindi, Alice sceglie $k = 10$ e calcola

  $$R = [10]G = (7, 11) \quad \text{e} \quad S = M + [10]B = (10, 11)$$

  Quindi, invia a Bob la coppia (R, S).
- Bob, per decriptare il messaggio calcola

  $$S - [3]R = (7, 11) - [3](10, 11) = (9, 16)$$

  che sa corrispondere alla parola "CIAO".

Sicurezza

Eve, conosce q, G, B e può intercettare la coppia (R, S). Per decriptare il messaggio ha bisogno o della chiave privata di Bob b oppure dell'elemento k. Infatti, conoscendo quest'ultimo, è semplice ricavarsi M a partire da S. In ogni caso, entrambi questi problemi richiedono di risolvere un ECDLP.

Anche in questa variante è molto importante che il valore di k sia distinto per ogni messaggio. Infatti, se Alice manda a Bob due messaggi M_1 ed M_2 con lo stesso k, allora Eve può calcolare $S_2 - S_1 = M_2 - M_1$. Quindi, conoscendo M_1 è possibile ricavarsi M_2 (o viceversa).

2.5.4 Elliptic Curve ElGamal Digital Signature

Per completezza, presentiamo il protocollo di firma basato sullo schema di ElGamal. Per la descrizione di questo algoritmo faremo uso di una funzione $f : E(\mathbb{F}_q) \to \mathbb{Z}$, che dato un punto $P = (x, y)$, ne restituisce un numero intero. Nelle applicazioni, tale mappa può essere pensata come la proiezione sulla prima coordinata, ovvero $f(P) = P_x = x$.

- *Scelta dei parametri:* Alice sceglie una curva ellittica E definita su un campo $\mathbb{F}_q$ in cui il problema dell' ECDLP sia computazionalmente difficile. Inoltre, sceglie un punto $G \in E(\mathbb{F}_q)$ di ordine un primo grande n.
- *Generazione delle chiavi:* A questo punto, Alice sceglie casualmente un intero $a \in \{2, \ldots, n-1\}$ e calcola $A = [a]G$. Inoltre, sceglie una funzione $f : E(\mathbb{F}_q) \to \mathbb{Z}$ che dato un punto sulla curva, ne restituisce la coordinata x. La chiave pubblica è (E, q, G, A, f).

- *Firma:* Alice vuole firmare il messaggio m codificato mediante un intero minore di n. Sceglie casualmente un intero $k \in \{2, \dots, n-1\}$ e calcola

$$R = [k]G \quad \text{e} \quad s \equiv k^{-1}(m - a \cdot f(R)) \bmod n$$

  Infine manda la tripla (m, R, s) a Bob.

- *Verifica:* Bob verifica l'identità del mittente calcolando

$$V_1 = [f(R)]A + [s]R \quad \text{e} \quad V_2 = [m]G$$

  Se $V_1 = V_2$ Bob conferma l'autenticità del messaggio.

La correttezza, si basa sul fatto che, da $s \equiv k^{-1}(m - af(R)) \bmod n$ segue $sk = m - af(R) + c \cdot n$ per un opportuno $c \in \mathbb{Z}$. Inoltre, ricordiamo che $[n]G = \mathcal{O}$ per ipotesi, da cui

$$V_1 = [f(R)]A + [s]R = [f(R)a]G + [sk]G = [f(R)a]G + [m - af(R) + cn]G = [m]G + \mathcal{O} = [m]G = V_2$$

Rispetto alla descrizione originale, il calcolo di s è svolto modulo un numero primo. Quindi, non ci sono eventuali problemi di invertibilità.

Vediamo, ora, un esempio di tale protocollo.

Esempio 10. Schema di firma ECES.

- Alice sceglie la curva $E : y^2 = x^3 + 2x + 2$ definita sul campo $\mathbb{F}_{17}$. Tale curva forma un gruppo ciclico di ordine $\#E = 19$. Il punto iniziale scelto è $G = (5, 1)$ il cui ordine n coincide con quello del gruppo.

- Alice sceglie $a = 3$ e calcola $A = [3]G = (10, 6)$. Sceglie la funzione f e pubblica (E, q, G, A, f).

- Alice vuole firmare il messaggio "CIAO" codificato mediante il numero intero 11. Quindi, sceglie $k = 10$ e calcola

$$R = [10]G = (7, 11) \quad \text{e} \quad s \equiv 2(11 - 3 \cdot 7) \bmod 19 = 18$$

  Infine invia a Bob (m, R, s).

- Con le informazioni ricevute Bob calcola

$$V_1 = [7](10, 6) + [18](7, 11) = (13, 10)$$

  e

$$V_2 = [11](5, 1) = (13, 10)$$

  e consta che effettivamente coincidono.

Sicurezza

Solo chi consce a e k può manomettere lo schema di firma digitale. Tuttavia, per calcolare a conoscendo solo G ed A bisogna risolvere un logaritmo discreto. Lo stesso dicasi per k. Notiamo che, usando quest'ultimo, Eve può risalire alla chiave privata di Alice semplicemente ricavandosi k dalla definizione di s; infatti, non sussistono problemi di invertibilità.

Come sempre, per non creare falle nel protocollo, è meglio usare k diversi ad ogni firma. In casi contrario, Eve potrebbe collezionare (m_1, R, s_1) e (m_2, R, s_2) e calcolare

$$(s_1 - s_2) \cdot k \equiv m_1 - m_2 \bmod n$$

da cui

$$k \equiv (m_1 - m_2) \cdot (s_1 - s_2)^{-1} \bmod n$$

con il quale si può risalire ad a dal s_1 o s_2.

Anche in questa versione, sarebbe buona cosa firmare l'impronta di m, cioè scegliere una funzione di hash h e firmare $h(m)$. In questo modo si precludono attacchi di falsificazione esistenziale.

2.5.5 Elliptic Curve Digital Signature Algorithm

La variante del DSA su curve ellittiche (ECDSA) è un protocollo di firma digitale ampiamente utilizzato in tutto il mondo. Per fare un esempio, esso è utilizzato come unico schema di firma dei Bitcoin.

Tale protocollo fu esposto da Vanstone nel 1992, e dal 2000 è diventato uno schema standard di firma digitale.

- *Scelta dei parametri:* Alice sceglie una curva ellittica E definita su un campo $\mathbb{F}_q$ in cui il problema dell' ECDLP sia computazionalmente difficile. Inoltre, sceglie un punto $G \in E(\mathbb{F}_q)$ di ordine un primo grande n (nella pratica n è di almeno 160 bit).

- *Generazione delle chiavi:* A questo punto, Alice sceglie casualmente un intero $a \in \{2, \dots, n-1\}$ e calcola $A = [a]G$. Inoltre, sceglie una funzione $f : E(\mathbb{F}_q) \to \mathbb{Z}$ che dato un punto sulla curva, ne restituisce la coordinata x. La chiave pubblica è (E, q, G, A, f).

- *Firma:* Alice vuole mandare un messaggio m. Prima di tutto ne calcola l'impronta mediante una funzione di hash h che restituisce un output della stessa lunghezza di n. A questo punto, sceglie casualmente $k \in \{2 \dots, n-1\}$ e calcola

  $$R = [k]G \quad \text{e} \quad s \equiv k^{-1}(h(m) + af(R)) \bmod n$$

  Se $f(R) \equiv 0 \bmod n$ oppure $s = 0$, allora bisogna ripetere la fase di firma.

  Infine, Alice invia a Bob (m, R, s).

- *Verifica:* Bob riceve (m, R, s) e per verificarne la provenienza calcola

  $$u_1 \equiv s^{-1} \cdot h(m) \bmod n \quad \text{e} \quad u_2 \equiv s^{-1} \cdot f(R) \bmod n$$

Infine, calcola

$$V = [u_1]G + [u_2]A$$

ed accetta la firma soltanto se $V = R$ ($V_x \equiv R_x \bmod n$).

Se la firma è stata generata correttamente, allora, da $s \equiv k^{-1}(h(m) + af(R)) \bmod n$ otteniamo

$$\begin{aligned} k &\equiv s^{-1}h(m) + af(R)s^{-1} \bmod n \\ &\equiv u_1 + au_2 \bmod n \end{aligned}$$

Dal momento che G genera un gruppo ciclico di ordine n, possiamo moltiplicare ambo i membri dell'equazione sopra ed ottenere

$$[k]G = [u_1 + au_2]G$$

Sfruttando ora, l'associatività del gruppo $E(\mathbb{F}_q)$, si ha

$$[k]G = [u_1]G + [u_2][a]G$$

ovvero,

$$[k]G = [u_1]g + [u_2]A$$

cioè $R = V$.

Vediamo un esempio .

Esempio 11. Schema di firma ECDSA.

- Alice sceglie la curva $E : y^2 = x^3 + 2x + 2$ definita sul campo $\mathbb{F}_{17}$. Tale curva forma un gruppo ciclico di ordine $\#E = 19$. Il punto iniziale scelto è $G = (5, 1)$ il cui ordine n coincide con quello del gruppo.
- Alice sceglie $a = 7$ e calcola $A = [7]G = (0, 6)$. Sceglie la funzione f e pubblica (E, q, G, A, f).
- Alice vuole firmare il messaggio "CIAO" la cui impronta è $h(\text{"CIAO"}) = 26$. Quindi, sceglie $k = 10$ e calcola

  $$R = [10]G = (7, 11) \quad \text{e} \quad s \equiv 2(26 + 7 \cdot 7) \bmod 19 = 17$$

  Infine invia a Bob (m, R, s).
- Con le informazioni ricevute Bob calcola $s^{-1} \equiv 9 \bmod 19$ con cui computa

  $$u_1 = 9 \cdot 26 \bmod 19 = 6$$

  e

  $$u_2 = 9 \cdot 7 \bmod 19 = 6$$

  Infine, calcola

  $$V = [6](5, 1) + [6](0, 6) = (7, 11) = R$$

  Essendo $V = R$ allora la firma è autentica.

Sicurezza

Per potersi spacciare per Alice, Eve avrebbe bisogno della chiave privata a o dell'intero k. Tuttavia, con la sola conoscenza di G ed A ed eventualmente R, s, Eve dovrebbe essere in grado di risolvere un logaritmo discreto.

La diversificazione dei k per ogni firma è essenziale, in caso contrario Eve potrebbe collezionare (m_1, R, s_1) ed (m_2, R, s_2). Quindi

$$(s_1 - s_2) \cdot k \equiv (h(m_1) - h(m_2)) \bmod n$$

da cui

$$k \equiv (h(m_1) - h(m_2))(s_1 - s_2)^{-1} \bmod n$$

Chiaramente $h(m_i)$ è noto poiché Eve conosce m_i ed h è di pubblico dominio.

Per quanto banale, quest'ultimo punto non sempre è stato rispettato. A titolo d'esempio riportiamo un fatto di cronaca risalente al 2010. Un gruppo di hacker violò migliaia di account Sony collegati alla piattaforma Playstation 3. Infatti, l'algoritmo di autenticazione, basato sul ECDSA, non era stato implementato correttamente, in particolare il valore k era statico, da cui la buona riuscita dell'attacco.

2.6 Crittografia simmetrica o pubblica?

Supponiamo di voler costruire un sistema di comunicazione capace di cifrare e decifrare i messaggi scambiati dai vari utenti. Quale tipologia di crittosistema è meglio utilizzare? Una a chiave pubblica, per esempio RSA, oppure una a chiave simmetrica, per esempio AES?

In realtà la risposta non è così immediata e molto dipende dal tipo di utilizzo del sistema che vogliamo creare.

La crittografia a chiave simmetrica è molto rapida in termini computazionali e non richiede chiavi di lunghezza spropositate per cifrare in modo sicuro un messaggio. Per esempio, AES lavora con soli 128 bit, sufficienti a garantire un livello di sicurezza classificato come SECRET. Tuttavia, ci sono degli svantaggi. In primo luogo gli utenti di un crittosistema simmetrico devono concordare a priori sulla chiave da utilizzare. Se tale informazione viene trasmessa attraverso canali insicuri, un malintenzionato potrebbe intercettarla. Qui sorge il secondo problema: cifratura e decifratura utilizzano la stessa chiave. Chi la conosce può leggere una intera conversazione tra due utenti.

I sistemi a chiave asimmetrica, per contro, sono decisamente lenti e richiedono chiavi di lunghezza importanti per raggiungere un livello di sicurezza paragonabile a quello di AES-128. Per esempio, RSA ed ElGamal richiedono chiavi da almeno 3072 bit. Tuttavia, i vantaggi sono notevoli. La crittografia asimmetrica si fonda su problemi matematici computazionalmente difficili da risolvere (DLP, ECDLP e fattorizzazione), per tanto è più difficile per un malintenzionato risalire alle chiavi private degli utenti. Inoltre, tali protocolli sono di facile implementazione e decisamente più flessibili da utilizzare all'interno di un sistema costituito da milioni di utenti. Infine, essi possono essere utilizzati per firmare documenti.

Sistemi basati sul DLP o ECDLP?

All'interno dei crittositemi a chiave pubblica c'è una ulteriore divisione: protocolli basati sul DLP e fattorizzazione di interi (DH, ElGamal, DSA, RSA), oppure quelli basati sul ECDLP (ECDH, ECEE, ECES, ECDSA).

I primi sono di più facile concezione ed implementazione e sono più veloci nel cifrare e decifrare messaggi a parità di chiave usata. Tuttavia, il problema del logaritmo discreto risulta più arduo da risolvere in ambito ellittico. Inoltre, per ottenere un livello di sicurezza di 128 bit per esempio, i sistemi basati sul ECDLP richiedono chiavi di lunghezza pari a 256 bit, contro i già citati 3072, con un conseguente vantaggio in termini di memoria usata e una (quasi) parificazione della velocità di esecuzione degli algoritmi.

Dunque, qualora siamo sicuri della corretta implementazione delle curve ellittiche nel nostro hardware e non dobbiamo inviare messaggi pesanti, i sistemi ECC risultano i più vantaggiosi in termini di velocità e sicurezza.

Sistemi ibridi

Se volgiamo comunicare velocemente, inviando quantità importanti di dati a discapito della sicurezza, possiamo usare i sistemi a chiave simmetrica. In tale ambito rientra, per esempio, la distribuzione di servizi streaming in diretta, come le competizioni sportive.

Se il nostro obbiettivo è quello di cifrare e inviare messaggi relativamente leggeri (in termini di byte) attraverso canali pubblici insicuri, allora la crittografia a chiave pubblica è quella che fa al caso nostro.

In conclusione, la scelta migliore potrebbe essere una pacifica coesistenza di entrambi i metodi all'interno di un ecosistema crittografico, i così detti *sistemi ibridi*. In effetti, due utenti potrebbero utilizzare la crittografia asimmetrica per scambiarsi un chiave segreta utilizzando Diffie-Hellman, tale valore, fungerà poi da chiave simmetrica per un sistema classico come AES. WhatsApp è un esempio di tale scenario [50].

Capitolo 3

Cleptografia: attacco SETUP

Supponiamo di installare sul nostro computer un software realizzato da una certa azienda. Quando utilizziamo tale programma, non sappiamo se e come informazioni sensibili, quali e-mail e password, siano trasmesse all'azienda creatrice. Questo è il rischio derivante dai cosiddetti sistemi *a scatola nera* (dall'inglese "black-box"), ovvero software e hardware di cui l'utente conosce solo input e output, ignorando la struttura interna. Non bisogna pensare che tali dispositivi siano poco utilizzati, anzi. Le SIM dei nostri telefoni possono essere considerati dei black-box. In generale tutte le Smart Card possono essere considerate dei dispositivi a scatola nera, quindi parliamo anche di bancomat, carte di credito e carte prepagate. Nel 1996, Adam Young e Moti Yung pubblicarono un articolo in cui sottolineavano i rischi che si incorrono ad utilizzare crittositemi black-box [56]. In particolare, i due crittografi introdussero un attacco chiamato *SETUP* (Secretly Embedded Trapdoor with Universal Protection) implementato all'interno del sistema, il quale ruba, in sordina, la chiave privata o altre informazioni segrete. Questi ed altri tipi di attacchi rientrano nella branca della *cleptografia* (dal greco κλέπτειν , "rubare" e εγραφία, "scrittura"), ovvero l'arte di carpire informazioni "silenziosamente" e in modo sicuro. Tale nome fu coniato dagli stessi Young e Yung nel 1997 [53].

Ovviamente, un attacco cleptografico può essere implementato anche in dispositivi che non sono a scatola nera. Tuttavia, tali attacchi sono facili da individuare e perciò non li tratteremo in questa tesi.

Nelle pagine seguenti descriveremo un attacco cleptografico che può essere implementato nei sistemi basati sul logaritmo discreto: il *Discrete Log Kleptogram*. Applicheremo tale meccanismo ai protocolli Diffie-Hellman, ElGamal e DSA fornendo per ciascuno una descrizione esaustiva di come un malintenzionato possa rubare la chiave privata di un utente, od estrapolare un parametro segreto. Il tutto sarà completato da esempi.

Prima però, cercheremo di capire meglio cosa è la cleptografia. Successivamente forniremo le definizioni base di SETUP.

3.1 Cleptografia

Come già accennato nell'introduzione di questo capitolo, la *cleptografia* è quella branca della criptovirologia che studia come rubare informazioni in modo sicuro e celato. Essa, può essere vista come una naturale estensione dei canali subliminali descritti da Gus Simmons [45] di cui presentiamo una veloce panoramica nell'appendice A.

Un attacco cleptografico è un attacco in cui un malintenzionato utilizza la crittografia asimmetrica per installare un *backdoor asimmetrica* nel dispositivo della vittima.

Definizione 3.1. Una *backdoor* è un metodo per aggirare la normale autenticazione di un sistema crittografico.

Una *backdoor asimmetrica* è una backdoor utilizzabile solo da chi la ha installata, anche dopo che questa venga scoperta.

Con una metafora, una backdoor è quella piccola porticina nascosta nella tana del Bianconiglio, che permette ad Eve (anche se sarebbe Alice) di entrare nel Paese delle Meraviglie. Se, inoltre, tale porta è asimmetrica, allora Eve e solo Eve possiede la chiave per aprirla e chiuderla, cosicché, anche se qualcuno dovesse trovarla, non vi può accedere.

Nei loro articoli, Yung e Young parlano di "cryptography against cryptography" [53]. In effetti, la backdoor costruita non è stata aggiunta a parte, affiancata al sistema crittografico, ma è invece incorporata direttamente all'interno del protocollo.

Possiamo pensare ad un attacco cleptografico come ad un *cavallo di Troia*, ovvero un programma contaminato che, una volta eseguito dall'utente, apre la backdoor all'attaccante.

Nella storia, l'informatico Edward Snowden confermò che l'NSA (l'agenzia di sicurezza degli Stati Uniti) implementò un tale backdoor nel Dual EC DRBG (in sostanza, un generatore di numeri pseudocasuali) il quale fu per ben sette anni uno dei protocolli standard per la generazione sicura di numeri pseudocasuali. La notizia, risalente al decennio scorso, fece scalpore, tanto da essere ripresa dal New York Times [34]. Per ulteriori informazioni, si può consultare il file originale scritto da Shumow e Ferguson nel 2007, che per primi sospettatarono della presenza di questa backdoor cleptografica [42].

3.2 Definizioni di SETUP

Nell'ambito degli attacchi cleptografici, Young e Yung progettarono il sistema SETUP (Secretly Embedded Trapdoor with Universal Protection). Uno degli aspetti fondamentali di questo meccanismo è che un malintenzionato può rubare la chiave privata di un utente senza che quest'ultimo possa accorgersene. Infatti, il furto non avviene direttamente, ma utilizzando parametri pubblici, rendendo l'attacco sicuro e celato alla vittima (secretly embedded trapdoor).

Inoltre, il SETUP è costruito in un modo tale da permettere al solo implementatore dell'attacco di estrapolare chiavi private, chiudendo l'accesso ad altri malintenzionati (universal protection).

Inizialmente, Young e Yung fornirono un'unica definizione di SETUP [56]. In seguito, questa definizione venne rinominata SETUP *regolare* e se ne aggiunsero altre due: SETUP *debole* e *forte* [53].

Il primo SETUP del 1996 è descritto informalmente come un algoritmo di cifratura basato su canali subliminali. La cifratura avviene per mezzo di una funzione PKCS denotata con la lettera E. Per PKCS (public-key cryptography standards) si intende un insieme di funzioni standard usate per la crittografia asimmetrica. In concreto, E è un funzione unidirezionale.

SETUP regolare

Definizione 3.2. Sia C un crittosistema black-box di pubblico dominio. Il meccanismo *SETUP regolare* è un algoritmo che modifica il sistema C per ottenerne un altro C' tale che:

1. L'input di C' coincide con le specifiche pubbliche dell'input di C.
2. Il sistema C' lavora efficacemente con la funzione di cifratura E (ed altre) dell'assalitore sita in C'.
3. La funzione di decifratura D dell'assalitore non è sita in C' ed è conosciuta dal solo attaccante.
4. L'output di C' coincide cone le specifiche pubbliche dell'output di C. Inoltre, C' contiene dei bit (della informazione segreta dell'utente) i quali sono facilmente ricavabili dall'attaccante, ma nascosti ad altri.
5. Gli output di C e C' sono polinomialmente indistinguibili a tutti eccetto che all'attaccante.
6. Anche se gli utenti vengono a sapere che nel loro sistema è presente un algoritmo SETUP (per esempio mediante ingegneria inversa su componenti a prova di manomissione), essi non sono comunque in grado di determinare le informazioni passate o predire quelle future.

Definizione 3.3. Sia C un crittosistema black-box di pubblico dominio. Un crittosistema *contaminato* C' è una versione modificata del sistema C contenente un meccanismo SETUP.

Cosa vogliono dire questi sei punti? Il primo assicura che un utente è capace di usare il sistema C' esattamente come usa C. Se gli input fossero diversi, l'utente potrebbe insospettirsi. La seconda condizione afferma che computazionalmente parlando i sistemi C e C' sono simili; ovvero se C rientra nella classe polinomiale, allora anche C' deve rientrare nella stessa classe. Il terzo punto asserisce che nessuno, al di fuori dell'attaccante può decriptare l'informazione (poiché solo l'attaccante possiede la funzione D). Affinché non ci siano perplessità da parte dell'utente, l'output di C e C' dovrebbero essere coerenti. Inoltre, dalle uscite di C', l'attaccante può estrapolare le informazioni della chiave segreta dell'utente, e questo era il quarto punto. La quinta condizione rafforza la precedente: oltre ad avere output simili, l'utente non deve essere in grado di trovarvi eventuali differenze mediante algoritmi con complessità polinomiale (cioè rapidi). Infine, l'ultimo punto dichiara che, se un utente dovesse sapere che il sistema è stato contaminato, non è tuttavia capace di (ri)calcolare le chiavi private passate o future (usando l'informazione estrapolata dal sistema contaminato).

SETUP debole

Definizione 3.4. Un *SETUP debole* è un SETUP regolare eccetto che per il quinto punto della definizione 3.2, il quale è sostituito da:

5 Gli output di C e C' sono polinomialmente indistinguibili a tutti eccetto che all'attaccante e al proprietario del dispositivo che ha il controllo della propria chiave privata.

Questa nuova condizione asserisce che non solo l'attaccante è capace di distinguere in tempo polinomiale gli output di C e C', ma anche il proprietario del dispositivo in quanto conoscitore di alcuni parametri privati.

Tale meccanismo potrebbe apparire a primo impatto insicuro. Tuttavia, un utente dovrebbe assumere, prima di tutto, che il device in questione contenga un algoritmo SETUP. In secondo luogo, deve essere in grado di identificarlo all'interno di un dispositivo black-box. Quindi, un SETUP debole può comunque essere utilizzato. Per esempio, Alice può contaminare lei stessa il crittosistema con un SETUP debole in modo da far trapelare la sua chiave privata a Bob. Dopodiché, i due disporranno di un canale subliminale con cui comunicare.

Già da questo semplice esempio si può intuire come il SETUP sia strettamente collegato ai canali subliminali anche se, in un certo senso, è migliore.

SETUP forte

Uno degli aspetti fondamentali del SETUP regolare è che l'utente non ha accesso all'effettiva implementazione del sistema C. Questa è una condizione necessaria per l'indistinguibilità polinomiale degli output di C e C'. Per la definizione di *SETUP forte* Young e Yung assumono che qualche volta viene usato il dispositivo contaminato, altre l'algoritmo sano.

Definizione 3.5. Un *SETUP forte* è un SETUP regolare con in aggiunta il seguente punto.

7 Gli utenti sono in grado di analizzare il dispositivo sia prima che dopo un eventuale uso. Inoltre, essi sono in grado di esaminare l'effettiva implementazione di C'. Tuttavia, gli utenti non possono riottenere le chiavi passate o predire quelle future. In più, se il meccanismo SETUP non dovesse essere sempre utilizzato in futuro, le chiavi sane e le chiavi manomesse resterebbero comunque polinomialmente indistinguibili.

Parafrasando, l'utente ha a disposizione ulteriori informazioni sul suo dispositivo: è in grado di applicare l'ingegneria inversa per conoscere gli algoritmi interni prima e dopo ogni utilizzo del device, ed è in grado di conoscere le funzioni e i parametri fissi usati in C'. Tuttavia, non conosce le chiavi private dell'assalitore o altri valori scelti casualmente dal dispositivo. Quindi, l'utente non è in grado di distinguere in tempi ragionevoli le informazioni sane da quelle contaminate. Infine, non è in grado di ricalcolare le sue chiavi, né di sapere quando viene utilizzato il SETUP e quando no.

Per questi motivi il SETUP forte è più sicuro dei SETUP debole e regolare. Per esempio, supponiamo che venga utilizzato il SETUP forte con probabilità del 50% (altrimenti il dispositivo funzionerebbe normalmente). Allora, l'utente non può dire con probabilità maggiore del 50% se o no l'output contiene informazioni nascoste sulla chiave segreta.

Poco più avanti descriveremo varie implementazioni del meccanismo SETUP in alcuni protocolli crittografici analizzati in precedenza. Per ogni esempio diremo se si tratterà di un SETUP debole, regolare oppure forte.

3.2.1 Schema di dispersione

Sempre in [53], i due autori presentano la definizione di *leakage bandwidth*) . Con tale termine si vuole indicare la quantità di informazioni di un utente che un malintenzionato riesce a ricavare mediante più iterazioni del dispositivo manomesso.

Definizione 3.6. Uno *schema di dispersione* (m, n) è un meccanismo SETUP che utilizza n chiavi/messaggi (output del dispositivo) per estrapolare m chiavi/messaggi ($m \leq n$) dell'utente.

Nel seguito useremo anche solo con (m, n) per denotare la leakage bandwidth di un dispositivo.

Per esemplificare, se un hacker ha bisogno di due chiavi pubbliche per ricavare una chiave privata, allora il sistema è $(1, 2)$.

3.2.2 Cleptogramma

Definizione 3.7. Un *cleptogramma* è un valore pubblicamente visibile che può essere un chiave pubblica, una firma, un testo cifrato ecc., tale da contenere al suo interno un valore nascosto, utilizzabile per violare un crittosistema.

Vedremo a breve, che nel SETUP implementato nello scambio di chiavi di Diffie-Hellman, la seconda chiave pubblica generata da Alice conterrà le informazioni utili ad Eve per impadronirsi della chiave privata della ragazza.

In sostanza, un cleptogramma non è altro che la chiave per aprire la backdoor installata grazie al SETUP.

3.3 Discrete Log Kleptogram

In [57], Young e Yung promossero un attacco cleptografico generale contro i protocolli basati sul problema del logaritmo discreto. Tale attacco prende il nome di *Discrete Log Kleptogram* (DLK). Ovviamente, esso rientra negli attacchi SETUP.

Il DLK lavora con due iterazioni del protocollo, andando a manomettere la generazione della seconda chiave privata in Diffie-Hellman, o dei k in ElGamal e DSA. Come? Vediamo quali sono i passi generali per eseguire questo attacco.

1. *Identificazione del cleptogramma*: conoscendo il protocollo sottostante utilizzato, Eve ricerca una potenza modulare $g^c \bmod p$ che sarà disponibile (o al più ricavabile) dagli output del dispositivo.

2. *Memorizzazione:* una volta stabilito il cleptogramma, il dispositivo esegue normalmente il protocollo crittografico, in particolare calcola $g^{c_1} \bmod p$. Quindi salva l'esponente c_1, in modo da essere disponibile durante il secondo utilizzo del dispositivo.

3. *Manomissione:* l'esponente precedentemente salvato è sfruttato, insieme alla chiave pubblica di Eve ed altri parametri, per generare un numero c_2 apparentemente casuale, ma compromesso. A questo punto, il dispositivo compie i passi originari, in particolare calcola e pubblica $g^{c_2} \bmod p$.

4. *Recupero:* Eve, collezionando $g^{c_2} \bmod p$ dagli output, riesce a ricavarsi c_2 mediante la sua chiave privata. Il valore $g^{c_2} \bmod p$ costituisce il vero cleptogramma.

Durante la fase di manomissione, l'esponente c_1 è utilizzato per generare quello che Young e Yung chiamano *elemento di campo nascosto*. Tale valore è inoltre dipendente da parametri installati dall'assalitore, e conosciuti solo da lui. Da questo elemento, sarà possibile costruire velatamente delle nuove chiavi private al secondo utilizzo del dispositivo. Queste, saranno facile preda dell'assalitore, poiché non sono casuali, ma dipenderanno dai suddetti parametri. Ma l'elemento di campo nascosto ha anche un'altra funzione; infatti, vedremo come giocherà un ruolo importante nella sicurezza del DLK, in particolare, sarà utile per celare correttamente il meccanismo SETUP agli occhi degli utenti.

Algoritmo

Entriamo nel dettaglio e presentiamo un algoritmo che descriva il DLK.

Supponiamo che Alice e Bob stiano comunicando con un certo protocollo crittografico basato sul logaritmo discreto. Supponiamo che tale dispositivo pubblichi solamente il valore $g^c \bmod p$, dove p è un numero intero, $c \in \{2, \cdots p-2\}$ e g è un generatore del gruppo G di ordine p.

Eve è riuscita ad impadronirsi del dispositivo di Alice e lo ha manomesso installando i seguenti dati: la chiave pubblica Y di Eve generata mediante la chiave privata X, tre interi fissati a, b e W, con quest'ultimo dispari, e una funzione di hash H crittograficamente sicura, che genera valore più piccoli di $\phi(p)$.

Adesso, possiamo descrivere le operazioni eseguite dal dispositivo contaminato di Alice.

- La prima volta che il dispositivo viene usato compie i seguenti passi:
  - Sceglie casualmente e in modo uniforme un intero $c_1 \in \{2, \ldots, p-2\}$.
  - Calcola e pubblica
    $$y_1 \equiv g^{c_1} \bmod p$$
  - c_1 è salvato in una memoria non volatile (il disco rigido ad esempio).
- La seconda volta che il dispositivo viene usato compie i seguenti passi:
  - Sceglie casualmente e in modo uniforme un intero $t \in \{0, 1\}$.
  - Calcola *l'elemento di campo nascosto*
    $$z \equiv g^{c_1 - Wt} Y^{-ac_1 - b} \bmod p$$
  - Calcola
    $$c_2 = H(z)$$
  - Calcola e pubblica
    $$y_2 \equiv g^{c_2} \bmod p$$

Adesso, conoscendo y_1 e y_2, Eve può risalire a c_2 compiendo quanto segue.

- Calcola
$$r \equiv y_1^a g^b \bmod p$$
- Usando la chiave privata X calcola
$$z_1 \equiv \frac{y_1}{r^X} \bmod p$$
- A questo punto ci sono due possibilità:
  - Se $y_2 \equiv g^{H(z_1)} \bmod p$, allora $H(z_1) = c_2$ e si conclude.
  - Altrimenti, calcola
$$z_2 \equiv \frac{z_1}{g^W}$$
e stavolta $y_2 \equiv g^{H(z_2)} \bmod p$. Allora $H(z_2) = c_2$ e si conclude.

Notiamo che, un eventuale intruso che voglia rubare c_2 dovrebbe conoscere la chiave privata X di Eve, ma questa non è interna al dispositivo e solo Eve ne è in possesso.

Possiamo notare come y_2 è il cleptogramma contente dei bit nascosti celanti l'informazione relativa a c_2. Mentre z è l'elemento di campo nascosto.

Ma questo metodo è davvero corretto? Sì e dipende dal valore casuale t usato dal dispositivo. Infatti, se $t = 0$, lavorando con z_1 si ha

$$\begin{aligned} z_1 &\equiv \frac{y_1}{r^X} \\ &\equiv \frac{g^{c_1}}{r^X} \\ &\equiv \frac{g^{c_1}}{(y_1^a g^b)^X} \\ &\equiv \frac{g^{c_1}}{(g^{ac_1+b})^X} \\ &\equiv g^{c_1}(g^{ac_1+b})^{-X} \\ &\equiv g^{c_1}(g^X)^{-ac_1-b} \\ &\equiv g^{c_1} Y^{-ac_1-b} \bmod p \end{aligned} \tag{3.1}$$

dove nell'ultima relazione si è usata la chiave pubblica di Eve $Y \equiv g^X \bmod p$. Notiamo che (3.1) coincide con z proprio con $t = 0$. Quindi $z_1 = z$, da cui $c_2 = H(z) = H(z_1)$.

Se, invece, il dispositivo lavora con $t = 1$, allora

$$\begin{aligned} z_2 &\equiv \frac{z_1}{g^W} \\ &\equiv g^{c_1} Y^{-ac_1-b} g^{-W} \\ &\equiv g^{c_1-W} Y^{-ac_1-b} \bmod p \end{aligned}$$

Ancora una volta abbiamo ritrovato z per $t = 1$. Quindi, $z_2 = z$ e $c_2 = H(z) = H(z_2)$.

Il ruolo di a, b, W e t

Come sottolineato dagli stessi autori, a, b, W sono interi inseriti a scopo precauzionale, nell'ipotetico caso in cui Alice sia in grado di invertire la funzione H.

Dunque, immaginiamo il seguente scenario. Alice, usando il suo dispositivo (che ricordiamo essere a scatola nera), si accorge che c'è qualcosa che non va e sospetta che Eve abbia implementato un meccanismo SETUP, in particolare il DLK. Alice, sa come questo attacco funziona e inoltre è in grado di invertire la funzione di hash H. Fortunatamente, Eve si è dimenticata implementare correttamente l'algoritmo ed ha imposto che il dispositivo lavora sempre con $t = 0$. Alice è in grado di capire se il suo dispositivo è stato contaminato? In effetti si, vediamo come.

Senza perdita di generalità supponiamo $a = 1$ e $b = 0$. Prima di tutto, Alice utilizza il dispositivo due volte, generando così, due coppie di chiavi privata-pubblica (c_1, y_1) e (c_2, y_2). Siccome conosce l'algoritmo SETUP, sa che $H(z) = c_2$ dove

$$z \equiv g^{c_1} Y^{-c_1}$$

Quindi, inverte H per calcolare $z = H(c_2)^{-1}$. Adesso, Alice calcola

$$f \equiv \frac{g^{c_1}}{z} \equiv \frac{g^{c_1}}{g^{c_1} Y^{-c_1}} \equiv Y^{c_1} \bmod p$$

Purtroppo, l'apparato utilizzato è black-box, quindi Alice non conosce il valore di Y. Tuttavia sa che $Y \equiv g^X \bmod p$. Dunque, inizia a fare delle supposizioni sull'intero X. Prima però, è necessario introdurre il concetto di residuo quadratico.

Definizione 3.8. Siano m ed n due un numeri interi. Allora m è detto *residuo quadratico* modulo n se esiste un intero x tale che

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

In caso contrario, m è detto *non-residuo quadratico* modulo n.

A questo punto, possiamo riprendere la discussione sulla natura di X.

- Se X fosse dispari, allora si aprirebbero due possibilità: c_1 o è pari o è dispari.
  - Se c_1 fosse pari, allora f si potrebbe riscrivere come

    $$f \equiv Y^{c_1} \equiv (Y)^{2c_1'} \equiv (Y^{c_1'}) \bmod p$$

    Ovvero, f sarebbe un residuo quadratico modulo p.
  - Se c_1 fosse dispari, allora f non sarebbe un residuo quadratico.
- Se X fosse pari , allora

  $$f \equiv Y^{c_1} \equiv (g^X)^{c_1} \equiv (g^2 X')^{c_1} = (g^{X' c_1})^2 \bmod p$$

  Ovvero f è un residuo quadratico modulo p.

Riassumendo quanto detto, Alice è in grado di capire se il suo sistema è stato contaminato oppure no semplicemente generando una enorme quantità di chiavi private c_i ed andando ad analizzare f sulle coppie (c_i, c_{i+1}). Se il dispositivo è sano, allora la quantità di residui quadratici è circa pari a quella dei non-residui quadratici; infatti, in $\mathbb{Z}_p^\times$ ci sono esattamente $(p-1)/2$ residui quadratici [25]. Se il dispositivo è contaminato, abbiamo appena dimostrato che la probabilità che f sia un residuo quadratico è del 75% contro il 25% che non lo sia, quindi ci aspetteremo di contare un maggior numeri di residui quadratici. Infine, ricordiamo che tale metodo si basa sul calcolare il simbolo di Legendre la cui complessità è $O(\log^3 p)$, dunque veloce [24].

Chiaramente, tale metodo è riproducibile qualora $W = 0$; infatti, in queste condizioni, il dispositivo lavorerebbe come con $t = 0$. Per tale motivo è importante la presenza congiunta di $W \neq 0$ e di t. Infine, notiamo che se Alice conoscesse esattamente quando il dispositivo utilizza $t = 0$, allora potrebbe collezionare tutte le chiavi generate ed analizzare i fattori f relativi.

Per ciò che concerne i valori a e b, noteremo che essi giocano un ruolo di maggior rilevanza negli algoritmi di firma digitale. Anticipiamo che essi sono inseriti come precauzione extra nel caso Alice riuscisse ad invertire la funzione H. Infatti, in tali schemi, Alice potrebbe essere in grado di calcolarsi c_1 e c_2 (non sono le chiavi private) a partire dalla sua chiave privata x_A. Se conoscesse a, b, Alice potrebbe calcolarsi z e confrontarlo con $H^{-1}(c_2)$ e rilevare la presenza del SETUP.

Sicurezza

Presentiamo ora la trattazione della sicurezza relativa all'implementazione dell'attacco SETUP DLK nei sistemi che si basano sul logaritmo discreto. I risultati che esporremo e dimostreremo in questa sezione valgono in tutta generalità, e possono essere facilmente adattati ai protocolli che in seguito andremo a trattare (DH, ElGamal e DSA).

Secondo Young e Yung, due sono le principali questioni legate alla sicurezza di tale SETUP.

1. Il meccanismo deve garantire che solo ed esclusivamente Eve può ricavare c_2, chiudendo le porte ad altri malintenzionati.

2. Nessuno, eccetto Eve, deve capire che un attacco SETUP è in esecuzione.

Per il primo problema, è richiesto che il calcolo di c_2 da parte dell'utente, o di altri hacker, deve essere difficile, anche avendo accesso alle specifiche del dispositivo usato, ovvero conoscendo Y, a, b, W. Infatti, ricordiamo che Alice non conosce le sue chiavi private c_1 e c_2, né tanto meno quella di Eve X.

A questo punto, Young e Yung affermano che:

Teorema 6. *L'attacco Discrete Log Kleptogram è sicuro se e solo se il Problema di Diffie-Hellman 2.6 è sicuro.*

Dimostrazione. $\Rightarrow$ Supponiamo esista una macchina A in grado di rompere l'attacco DLK (tale macchina è detta "oracle" in inglese). Questo vuole dire che, conoscendo Y e y_1, l'oracolo A calcola z_1 per $t = 0$ e z_2 per $t = 1$, da cui si potrà ricavare la chiave c_2. Denotiamo questo fatto con

$$A(Y, y_1) = (z_1, z_2)$$

Ciò che dobbiamo dimostrare è che, dati g^x e g^y, l'oracolo riesce a calcolare g^{xy}.

Ora, ricordando che $z_1 \equiv g^{c_1} Y^{-ac_1-b} \bmod p$, $z_2 \equiv g^{c_1-W} Y^{-ac_1-b} \bmod p$ e $y_1 = g^{c_1} \bmod p$, si ha

$$A(Y, g^{c_1}) = (g^{c_1} Y^{-ac_1-b}, g^{c_1-W} Y^{-ac_1-b})$$

A questo punto inseriamo g^x e g^y nella macchina A.

$$A(g^y, g^x) = (g^x (g^y)^{-ax-b}, g^{x-W} (g^y)^{-ax-b})$$

Usando la prima componente dell'output di A è possibile calcolare g^{xy}. Infatti, definiamo f come

$$f = \frac{g^x (g^y)^{-b}}{z_1} = \frac{g^x (g^y)^{-b}}{g^x (g^y)^{-ax-b}} = \frac{g^x (g^y)^{-b}}{g^x (g^y)^{-b} g^{-axy}} = (g^{xy})^a$$

da cui

$$g^{xy} = f^{1/a}$$

$\Leftarrow$ Vicevrsa, supponiamo esista un oracolo B capace di risolvere il DHP, ovvero, dati g^x e g^y, B può calcolare g^{xy}. Come prima, denotiamo questo fatto con

$$B(g^x, g^y) = g^{xy}$$

Dimostriamo che, dati Y, a, b, W, g, y_1 e H la macchina B è in grado di risalire a z, con cui si può calcolare $c_2 = H(z)$.

Definiamo f come

$$f = \frac{g^{c_1}}{B(g^{ac_1+b}, Y)}$$

dove

$$B(g^{ac_1+b}, Y) = g^{(ac_1+b)X} = (g^X)^{ac_1+b} = Y^{ac_1+b}$$

Mettendo insieme le due cose otteniamo

$$f = g^{c_1} Y^{-ac_1-b}$$

Questa scrittura coincide con z per $t = 0$. Dunque, per tale valore $f = z$. Altrimenti, se $t = 1$, allora $f/g^W = z$ (ricordiamo che W è noto). Usando la funzione di hash H, B calcola c_2.

□

Dunque, Alice (o un altro utente) non è in grado di estrapolare c_2 da ciò che è riuscita a conoscere sul dispositivo utilizzato. Per farlo, dovrebbe risolvere un DHP che sappiamo essere un problema di difficoltà analoga a quella del logaritmo discreto.

Consideriamo, ora, il secondo problema. Alice non deve essere in grado di capire se un attacco SETUP è in atto semplicemente guardando gli output. In primo luogo, notiamo che in un sistema sano, c_2 è scelto casualmente, quindi tutti i numeri da 2 a $p-2$ hanno equa probabilità di essere scelti. Se il sistema fosse contaminato, tale

casualità si perderebbe poiché c_2 sarebbe determinato da altri fattori. Quindi, qualcuno potrebbe accorgersi che c'è qualcosa che non quadra nel dispositivo. Ricordiamo che $c_2 = H(z)$. Dunque, affinché c_2 appaia casualmente scelto, $H(z)$ deve essere uniformemente distribuito in $\{1, \ldots, p-2\}$. Un primo passo per ottenere questo risultato è mostrare che z è uniformemente distribuito in $\mathbb{Z}_p^\times$.

Young e Yung assumono che $g_1 \equiv g^{1-Xa} \bmod p$ sia un generatore di $\mathbb{Z}_p^\times$. Sotto queste condizioni, dimostrano che

Teorema 7. *Supponendo che g_1 sia un generatore di $\mathbb{Z}_p^\times$, allora l'elemento di campo nascosto z è uniformemente distribuito in $\mathbb{Z}_p^\times$.*

Dimostrazione. Ricordiamo dall'espressione originaria di z che

$$z \equiv g^{c_1 - Wt} Y^{-ac_1 - b} \bmod p$$

da cui

$$\begin{aligned} z &\equiv g^{c_1 - Wt} Y^{-ac_1 - b} \\ &\equiv g^{c_1 - Wt} g^{-Xac_1 - Xb} \\ &\equiv g^{(1-Xa)c_1} \\ &\equiv g^{-Xb - Wt} g_1^{c_1} \bmod p \end{aligned} \tag{3.2}$$

Per ipotesi g_1 è un generatore, in particolare

$$g^{-Xb-Wt} \equiv g_1^u \bmod p$$

per un qualche intero $u \in \{1, \ldots, p-1\}$. Sostituendo a z si ha

$$z \equiv g_1^{u+c_1} \bmod p$$

Ma $c_1 \in \mathbb{Z}_p^\times$ è scelto casualmente, ciò prova che z è uniformemente distribuito in $\mathbb{Z}_p^\times$. □

Fermiamoci un momento ad analizzare il teorema precedente. Affinché sia valido, è necessario che g_1 sia un generatore del gruppo $\mathbb{Z}_p^\times$. Ma in concreto, quando ciò accade? Chiaramente Eve può scegliere ad hoc a ad X; tuttavia, questi sono valori fissati una volta per tutte, quindi, al variare di p, non è detto che g_1 sia ancora un generatore del gruppo.

Dunque non resta che accontentarci di una soluzione probabilistica del problema. Ovvero, ci chiediamo per quali valori di p, l'elemento g_1 ha maggior probabilità di essere un generatore del gruppo $\mathbb{Z}_p^\times$. Ricordiamo che $\mathbb{Z}_p^\times$ è gruppo ciclico di ordine $p-1$ i cui generatori sono tutti gli elementi minori di $p-1$ e coprimi con esso. In totale abbiamo $\phi(p-1)$ generatori. Ora, se scegliamo p in modo tale che $p-1$ contenga un fattore primo molto grande, allora il gioco è fatto. Tra queste possibilità di scelta rientrano quelle dei numeri primi detti *sicuri*.

Definizione 3.9. Un numero primo p si dice *sicuro* se è esprimibile nella forma $p = 2q+1$ dove q è un altro numero primo.

Supponendo che il p utilizzato da dispositivo sia un primo sicuro, allora $p = 2q - 1$, quindi $p - 1 = 2q$, ovvero $p - 1$ ha un fattore primo molto grande, q. Dunque $\phi(p-1) = \phi(2q) = \phi(2)\phi(q) = q - 1$. Infine, la probabilità che g_1 sia un generatore è

$$P = \frac{q-1}{p-1} = \frac{q-1}{2q} \approx \frac{1}{2} = 50\%$$

La probabilità non è altissima, tuttavia è molto più alta di quella che si avrebbe utilizzando primi non sicuri. Infatti, per $p = 2027 = 2 \cdot 1013 + 1$ allora $P \approx 0.499$. Mentre per $p = 2^{255} - 19$, $P \approx 0.33$.

Nel caso in cui $p = 2q + 1$ sia un primo sicuro e non cambi durante diversi utilizzi del dispositivo, allora abbiamo già detto che Eve può scegliere a ed X ad hoc affinché g_1 sia un generatore di $\mathbb{Z}_p^\times$. Infatti, se d è un numero dispari minore di q, allora $\gcd(d, p-1) = \gcd(d, 2q) = 1$ poiché i divisori di $2q$ sono $\{1, 2, q, 2q\}$. Quindi g^d genera tutto $\mathbb{Z}_p^\times$. Una volta trovato un tale d, Eve sceglie X ed a affinché $1 - Xa = d$.

A questo punto il gioco è quasi fatto. Sotto opportune ipotesi, si può dimostrare che la funzione H è uniformemente distribuita e che quindi gli output dei sistemi C e C' (sano e contaminato) sono polinomialmente indistinguibili.

Corollario 1. *Supponiamo che H sia una funzione pseudocasuale e che le specifiche del dispositivo utilizzato siano di pubblicamente consultabili. Allora gli output di C e C' sono polinomialmente indistinguibili.*

Dimostrazione. Dal teorema 7 sappiamo che z ha distribuzione uniforme in $\mathbb{Z}_p^\times$. Poiché H è una funzione pseudocasuale A.5, allora $c_2 = H(z)$ è uniformemente distribuito in $\mathbb{Z}_{p-1}$ (ricordiamo che H genera valori più piccoli di $p-1$). Dunque, i valori g^{c_2} emessi da C e C' hanno distribuzioni di probabilità polinomialmente indistinguibili. □

Ricapitoliamo quanto evidenziato in questa sezione. Detti C e C' rispettivamente il dispositivo sano e quello contaminato, allora:

1. L'input di C e C' coincidono, poiché entrambi i dispositivi lavorano con p e g.

2. Per calcolare la seconda chiave c_2 a partire c_1, il dispositivo C' esegue essenzialmente delle esponenziazioni, il cui costo computazionale è relativamente basso. Per cui, la funzione di cifratura E di Eve calcola in modo efficiente i valori necessari per un corretto attacco SETUP.

3. Per decifrare, Eve usa la sua chiave privata X che non è insita nel dispositivo contaminato.

4. L'output finale di C' contiene un cleptogramma, eppure esso appare coerente con ciò che C dovrebbe produrre.

5. Il corollario 1 asserisce che gli output di C e C' sono polinomialmente indistinguibili.

6. Il teorema 6 afferma che, anche qualora Alice riuscisse a scoprire la presenza di un meccanismo SETUP nel suo dispositivo, sarebbe impossibile estrapolare la chiave c_2. Quindi, Alice non sarebbe in grado di ri-calcolarsi le chiavi prodotte in passato (o anche quelle create in futuro).

7. Sempre per teorema 6 e per quanto discusso in 3.3, anche qualora Alice fosse in grado di conoscere tutti i parametri dell'attacco SETUP implementato nel suo dispositivo, sarebbe impossibile riottenere le chiavi passate e future. Infine, ancora il corollario 1 asserisce che, nel caso in cui il SETUP non dovesse essere sempre utilizzato, gli output sani e quelli contaminati resterebbero comunque polinomialmente indistinguibili.

Confrontando questa lista con la Definizione 3.5 ci accorgiamo che il meccanismo DLK implementato costituisce un SETUP forte. Inoltre, esso è uno schema $(1, 2)$ poiché servono due iterazioni del dispositivo di Alice per poter estrapolare c_2.

Notiamo che tale schema può essere, in realtà, esteso ad uno del tipo $(m, m+1)$. Infatti, supponendo che Alice voglia comunicare anche con Dave, la nuova chiave privata c_3 non sarebbe casuale, ma

$$c_3 = H(z)$$

dove

$$z \equiv g^{c_2 - Wt} Y^{-ac_2 - b} \bmod p$$

Eseguendo i desueti passi, Eve può così ricavarsi anche c_3. Iterando il processo, otteniamo uno schema $(m, m+1)$.

Concludiamo la sezione relativa al DLK con una piccola precauzione.

Osservazione 5. Nel seguito descriveremo gli attacchi SETUP in DH, ElGamal e DSA. Essenzialmente essi sono dei DLK implementati in opportune parti del protocollo base. Tuttavia, questo non vuol dire che vengono ereditati tutti i fattori relativi alla sicurezza. Molto dipende dal protocollo sottostante e in quale parte si inserisce il DLK. A titolo d'esempio, anticipiamo fin da subito che lo schema di firma di ElGamal e il DSA non sono dei SETUP forti, ma solo regolari.

3.3.1 SETUP nello scambio di chiavi di Diffie-Hellman

Il SETUP implementato nello scambio di chiavi di Diffie-Hellamn ricalca passo per passo l'algoritmo DLK appena descritto. Tuttavia, riteniamo giusto riportare per completezza tale schema.

Ricordiamo che se Alice e Bob vogliono comunicare con un sistema simmetrico, prima devono generare la chiave. Decidono di usare lo scambio di Diffie-Hellamn. Quindi, entrambi scelgono un primo p grande e un generatore g del gruppo $\mathbb{Z}_p^\times$. Quindi, generano le rispettive chiavi private x_A e x_B in $\{2, \ldots, p-2\}$ e trasmettono le chiavi pubbliche $k_A \equiv g^{x_A} \bmod p$ e $k_B \equiv g^{x_B} \bmod p$.

Supponiamo che il dispositivo di Alice sia stato contaminato con un meccanismo SETUP da Eve. Il device di Alice si assume essere black-box, dunque la ragazza non è in grado di conoscere la sua chiave segreta. Inoltre, affinché Eve possa estrapolare la chiave privata di Alice, quest'ultima deve generare due chiavi segrete. Dunque supponiamo che Alice effettui due scambi di chiavi alla Diffie-Hellman: uno con Bob e un altro con un nuovo amico Carol, per cui dovrà generare una seconda chiave privata.

Algoritmo

Alice usa il protocollo DH con Bob e Carol. Dunque, genera due chiavi private che denoteremo rispettivamente con x_1 e x_2. I parametri p e g non variano. Eve è riuscita ad impadronirsi del dispositivo di Alice e lo ha manomesso installando un meccanismo SETUP che consta dei seguenti dati: la chiave pubblica di Eve Y, tre interi fissati a, b e W, con quest'ultimo dispari, e una funzione hash H crittograficamente sicura che genera valori più piccoli di $p - 1$.

A questo punto, il dispositivo di Alice compie quanto segue:

- La prima volta che il dispositivo viene usato, cioè trasmette come output la prima chiave pubblica, esso esegue:
  - Il dispositivo sceglie uniformemente a caso un intero $x_1 \in \{2, \ldots, p-2\}$ (la chiave privata di Alice).
  - Calcola e pubblica
$$k_1 \equiv g^{x_1} \bmod p$$
  - x_1 è salvata in una memoria non volatile (disco rigido) in modo da essere utilizzata al secondo utilizzo dell'apparato.
- La seconda volta che il dispositivo viene usato, esso esegue:
  - Il dispositivo sceglie uniformemente a caso un intero $t \in \{0, 1\}$.
  - Calcola
$$z \equiv g^{x_1 - Wt} \cdot Y^{-ax_1 - b} \bmod p$$
  - Calcola la seconda chiave privata di Alice nel seguente modo
$$x_2 = H(z)$$
  - Calcola e pubblica
$$k_2 \equiv g^{x_2} \bmod p$$

A questo punto, Eve non deve fare altro che collezionare le due chiavi pubbliche k_1 e k_2. Dopodiché, è in grado di recuperare x_2 nel seguente modo.

- Eve calcola
$$r \equiv k_1^a g^b \bmod p$$
- Usando la sua chiave privata X calcola
$$z_1 \equiv \frac{k_1}{r^X} \bmod p$$
- A questo punto ci sono due possibilità.
  - Se $k_2 \equiv g^{H(z_1)} \bmod p$, allora $H(z_1) = x_2$ e si conclude.

– Altrimenti, Eve calcola

$$z_2 = \frac{z_1}{g^W}$$

e stavolta $k_2 \equiv g^{H(z_2)} \bmod p$. Allora $H(z_2) = x_2$ e si conclude.

Non entreremo nei dettagli della verifica della correttezza e del ruolo di a, b, W. Essi sono del tutto analoghi a i già studiati per il DLK.

Vediamo un esempio concreto del funzionamento di questo attacco.

Esempio 12.

- Per prima cosa, Eve genera la sua coppia di chiavi. Per farlo ha bisogno di p e di g. Supponiamo che $p = 151$ e $g = 6$. A questo punto, Eve sceglie $X = 51$ e calcola la sua chiave pubblica

$$Y \equiv g^X \equiv 41 \bmod 151$$

- Alice vuole comunicare con Bob e con Carol, prima di tutto genera la coppia di chiavi da usare con Bob. Quindi sceglie

$$c_1 = 123$$

da cui può calcolare la chiave pubblica

$$k_1 \equiv g^{x_1} \equiv 83 \bmod 151$$

Quindi invia k_1 a Bob. Quest'ultimo seguirà la classica routine per condividere la chiave segreta con Alice.

- Per comunicare con Carol, Alice deve calcolare un nuovo paio di chiavi. Tuttavia, Eve si era preventivamente impossessata del dispositivo di Alice, in cui ha installato Y e gli interi $a = 1, b = 4, W = 111$. Supponiamo inoltre che $t = 0$. Quindi, Alice chiede di generare la chiave segreta, ma il dispositivo calcola

$$\begin{aligned} z &\equiv g^{x_1 - Wt} Y^{-ax_1 - b} \\ &\equiv 6^{123 - 111 \cdot 0} 41^{-1 \cdot 123 - 4} \\ &\equiv 6^{123} 41^{-127} \\ &\equiv 94 \bmod 151 \end{aligned}$$

A questo punto, con la funzione H, il dispositivo produce la seconda chiave

$$x_2 = H(z) = H(94) = 26$$

da cui la chiave pubblica

$$k_2 \equiv g^{x_2} \equiv 47 \bmod 151$$

Infine, Alice manda k_2 a Carol.

- Carol calcola la sua coppia di chiavi

$$x_C = 8 \quad \text{e} \quad k_C \equiv g^{x_C} \equiv 43 \bmod 151$$

ed invia ad Alice k_C.

- A questo punto, Alice e Carol calcolano la chiave segreta

$$K \equiv k_C^{x_2} \equiv k_2^{x_C} \equiv 17 \bmod 151$$

- Ora Eve può portare a termine il suo attacco. Colleziona k_1 ed k_2, con cui recupera x_2: prima di tutto calcola

$$r \equiv k_1^a g^b \equiv 83^1 6^4 \equiv 56 \bmod 151$$

Successivamente calcola

$$z_1 \equiv \frac{k_1}{r^X} \equiv \frac{83}{56^{51}} \equiv 94 \bmod 151$$

Ora, Eve calcola

$$g^{H(z_1)} \equiv 6^{H(94)} \equiv 47 \bmod 151$$

e nota che coincide con k_2. Quindi conclude che $z_1 = z$, ma sopratutto

$$x_2 = H(z) = H(z_1) = 26$$

Infine, usando la chiave pubblica k_C di Carol, Eve può calcolare K

$$K \equiv k_C^{x_2} \equiv 17 \bmod 151$$

con cui può decriptare i messaggi scambiati tra Alice e Carol.

Sicurezza

Come anticipato all'inizio della sezione, tale SETUP ripercorre esattamente il DLK, quindi, senza spendere ulteriori parole, possiamo affermare che il SETUP implementato nello scambio di chiavi di Diffie-Hellman è un SETUP forte. Inoltre esse è uno schema $(1, 2)$ generalizzabile ad uno schema $(m, m + 1)$.

3.3.2 SETUP nello schema di cifratura di ElGamal

In 2.4.3, abbiamo visto come il crittografo egiziano ElGamal abbia costruito due protocolli crittografici. Il primo riguarda la cifratura di un messaggio, mentre il secondo affronta il problema della firma digitale.

Ciò che ci proponiamo di fare ora, è quello di vedere come il DLK possa essere implementato nei sistemi di ElGamal, sia di firma digitale, sia di cifratura. Iniziamo da quest'ultimo.

In prima analisi, notiamo che ci sono due possibili cleptogrammi da sfruttare. Il primo si cela nella fase iniziale di generazione delle chiavi. Tuttavia, una eventuale implementazione del SETUP non sarebbe plausibile poiché richiederebbe la generazione di più chiavi private da parte di uno stesso utente. Nella pratica questo non accade, ovvero, Alice e Bob, una volta generate le rispettive chiavi private, useranno sempre e solo quelle per cifrare e decifrare uno o più messaggi. Tuttavia, un SETUP implementato in questa fase (diverso dal DLK) fu inizialmente proposto da Yung e Young in [56] e per una descrizione dettagliata del meccanismo si può consultare [2].

Il secondo sentiero consiste nel manomettere il messaggio. In particolare, è possibile realizzare un cleptogramma sfruttando il k utilizzato nel protocollo durante la fase di cifratura. Un DLK implementato in questa sezione dell'algoritmo si rivela molto più efficace e concreto rispetto al primo. Per tale motivo, ci limiteremo a trattare solo questo secondo attacco.

Algoritmo

Supponiamo che Alice voglia mandare un paio di messaggi M_1, M_2 a Bob utilizzando il protocollo di ElGamal. Per far ciò, Alice acquisisce la chiave pubblica di Bob (p, g, k_B) dove p e g sono parametri fissi. Eve è riuscita ad impadronirsi del dispositivo di Alice e lo ha manomesso installando un meccanismo SETUP che consta dei seguenti dati: la chiave pubblica di Eve Y, tre interi fissati a, b, W, con quest'ultimo dispari, e una funzione di hash H crittograficamente sicura, che genera valori più piccoli di $p-1$.

A questo punto il dispositivo di Alice compie quanto segue.

- La prima volta che il dispositivo viene usato, cioè Alice manda il primo messaggio M_1 a Bob, esegue.
  - Il dispositivo, sceglie uniformemente a caso un intero $k_1 \in \{2, \ldots, p-2\}$.
  - Cifra il messaggio M_1 secondo ElGamal, dunque calcola
  $$r_1 \equiv g^{k_1} \bmod p \quad \text{e} \quad S_1 \equiv M_1 k_B^{k_1} \bmod p$$
  - La coppia (r_1, S_1) è inviata a Bob, il quale la sfrutta per decifrare il messaggio ed ottenere M_1. Inoltre, k_1 è salvata in una memoria non volatile.
- La seconda volta che il dispositivo viene usato, cioè per mandare un secondo messaggio M_2, esso esegue.
  - Il dispositivo sceglie uniformemente a caso un intero $t \in \{0, 1\}$.
  - Calcola
  $$z \equiv g^{k_1 - Wt} \cdot Y^{-ak_1 - b} \bmod p$$
  - Calcola
  $$k_2 = H(z)$$

– Cifra il messaggio M_2 secondo ElGamal, dunque calcola

$$r_2 \equiv g^{k_2} \bmod p \quad \text{e} \quad S_2 \equiv M_2 k_B^{k_2} \bmod p$$

– La coppia (r_2, S_2) è inviata a Bob, il quale la sfrutta per decifrare il messaggio ed ottenere M_2.

A questo punto, Eve non deve fare altro che collezionare le coppie (r_i, S_i). Dopodiché, è in grado di recuperare il messaggio M_2 nel seguente modo.

- Eve calcola
$$R \equiv r_1^a g^b \bmod p$$
- Usando la sua chiave privata X calcola
$$z_1 \equiv \frac{r_1}{R^X} \bmod p$$
- A questo punto ci sono due possibilità.
  – Se $r_2 \equiv g^{H(z_1)}$, allora $H(z_1) = k_2$.
  – Altrimenti, Eve calcola
  $$z_2 \equiv \frac{z_1}{g^W}$$
  e stavolta $r_2 \equiv g^{H(z_2)} \bmod p$. Allora $H(z_2) = k_2$.
- In entrambi i casi, Eve riesce a recuperare k_2, a questo punto, non deve fare altro che calcolare
$$M_2 \equiv \frac{S_2}{k_B^{k_2}} \bmod p$$

Quindi, Eve è in grado di decriptare il secondo messaggio M_2 senza collezionare nessuna chiave privata, ma semplicemente violando la casualità di k_2 imponendo che sia determinato dai fattori interni installati da Eve. Quindi, $r_2 \equiv g^{k_2} \bmod p$ funge da cleptogramma in questo SETUP.

Non entreremo nei dettagli della verifica della correttezza e del ruolo di a, b, W. Essi sono del tutto analoghi a quelli già studiati per il DLK.

Vediamo un esempio concreto del funzionamento di questo attacco.

Esempio 13.
- Per prima cosa, Eve genera la sua coppia di chiavi. Per farlo ha bisogno di p e di g. Supponiamo che $p = 151$ e $g = 6$. A questo punto, Eve sceglie $X = 51$ e calcola la sua chiave pubblica
$$Y \equiv g^X \equiv 41 \bmod 151$$
- Alice vuole comunicare con Bob. Prima di tutto scarica la chiave pubblica di Bob k_B. Supponiamo $x_B = 18$ da cui $k_B \equiv 124 \bmod 151$.

- A questo punto, Alice vuole inviare il messaggio M_1 ="SONO" a Bob. Il primo passo è la codifica di M_1 tramite un numero intero, supponiamo $M_1 = 147$. Quindi, Alice sceglie
$$k_1 = 123$$
con cui può calcolare
$$r_1 \equiv g^{k_1} \equiv 83 \bmod 151$$
ed
$$S_1 \equiv M_1 k_B^{k_1} \equiv 147 \cdot 124^{123} \equiv 35 \bmod 151$$
Quindi invia la coppia (r_1, S_1) a Bob. Quest'ultimo seguirà la classica routine per condividere decifrare il messaggio.

- Per inviare il secondo messaggio M_2 ="IO", codificato come $M_2 = 78$, Alice deve calcolare una nuova coppia (r_2, S_2). Tuttavia, Eve si era preventivamente impossessata del dispositivo di Alice, in cui ha installato Y e gli interi $a = 1, b = 4, W = 111$. Supponiamo inoltre che $t = 0$. Quindi, Alice chiede di generare la chiave segreta, ma il dispositivo calcola
$$\begin{aligned} z &\equiv g^{k_1 - Wt} Y^{-ak_1 - b} \\ &\equiv 6^{123 - 111 \cdot 0} 41^{-1 \cdot 123 - 4} \\ &\equiv 6^{123} 41^{-127} \\ &\equiv 94 \bmod 151 \end{aligned}$$
A questo punto, con la funzione H, il dispositivo produce la seconda chiave
$$k_2 = H(z) = H(94) = 26$$
con cui calcola
$$r_2 \equiv g^{k_2} \equiv 47 \bmod 151$$
e
$$S_2 \equiv M_2 k_B^{k_2} \bmod p \equiv 78 \cdot 124^{26} \equiv 8 \bmod 151$$
Il tutto è inviato a Bob

- Ora Eve può portare a termine il suo attacco. Colleziona (r_1, S_1) ed (r_2, S_2), poi calcola
$$R \equiv r_1^a g^b \equiv 83^1 6^4 \equiv 56 \bmod 151$$
Successivamente calcola
$$z_1 \equiv \frac{r_1}{R^X} \equiv \frac{83}{56^{51}} \equiv 94 \bmod 151$$
Ora, Eve calcola
$$g^{H(z_1)} \equiv 6^{H(94)} \equiv 47 \bmod 151$$
e nota che coincide con r_2. Quindi conclude che $z_1 = z$, ma sopratutto
$$k_2 = H(z) = H(z_1) = 26$$

Infine, usando la chiave pubblica k_B di Bob e k_2 appena recuperato, Eve può calcolare M_2

$$M_2 \equiv \frac{S_2}{k_B^{k_2}} \equiv \frac{8}{124^{26}} \equiv 78 \bmod 151$$

Adesso Eve sa che il secondo messaggio ad essere stato inviato era "IO".

Sicurezza

Ricordiamo che il DLK è implementabile allo stesso modo in tutti i sistemi basati sul logaritmo discreto esattamente come fatto per lo scambio DH. Quindi, possiamo concludere che l'attacco SETUP implementato nel sistema di cifratura di ElGamal è un SETUP forte ed è uno schema $(1, 2)$. Sulla falsa riga di quanto già visto per il DH, possiamo estendere l'attacco ad un meccanismo $(m, m + 1)$.

3.3.3 SETUP nello schema di Firma Digitale di ElGamal

Lo schema di firma digitale non si discosta molto da quello di cifratura, quindi non dovranno sorprendere eventuali somiglianze nella descrizione dei due attacchi.

In primo luogo, notiamo che Young e Yung [56] proposero un attacco SETUP, diverso dal DLK, capace di far trapelare la chiave privata di un utente e implementato durante la generazione delle chiavi. Tuttavia, questo si rivela essere un SETUP debole e quindi non molto sicuro dal punto di vista di Eve. In questa sezione, invece, discuteremo il DLK applicabile all'elemento k generato casualmente durante la fase di firma. Per chi fosse interessato al primo approccio, consigliamo nuovamente la lettura di [2].

Algoritmo

Supponiamo che Alice voglia mandare un paio di messaggi M_1, M_2 a Bob e firmarli, utilizzando il relativo protocollo di ElGamal. Per far ciò, Alice acquisisce la chiave pubblica di Bob (p, g, k_B) dove p e g sono parametri fissi. Eve è riuscita ad impadronirsi del dispositivo di Alice e lo ha manomesso installando un meccanismo SETUP che consta dei seguenti dati: la chiave pubblica di Eve Y, tre interi fissati a, b, W, con quest'ultimo dispari, e una funzione di hash H crittograficamente sicura, che genera valori più piccoli di $p - 1$.

A questo punto il dispositivo di Alice compie quanto segue.

- La prima volta che il dispositivo viene usato, cioè Alice manda il primo messaggio M_1 a Bob, esegue.
  - Il dispositivo, sceglie uniformemente a caso un intero $k_1 \in \{2, \ldots, p - 2\}$ tale che $\gcd(k, p - 1) = 1$.
  - Per firmare il messaggio M_1, il dispositivo calcola

$$r_1 \equiv g^{k_1} \bmod p \quad \text{e} \quad s_1 \equiv k_1^{-1}(M_1 - xr_1) \bmod p - 1$$

- La coppia (r_1, S_1) e il messaggio M_1 sono inviati a Bob, il quale verifica l'autenticità della firma. Inoltre, k_1 è salvata in una memoria non volatile.

- La seconda volta che il dispositivo viene usato, cioè per mandare un secondo messaggio M_2, esso esegue.

  - Il dispositivo sceglie uniformemente a caso un intero $t \in \{0, 1\}$.
  - Calcola
  $$z \equiv g^{k_1 - Wt} \cdot Y^{-ak_1 - b} \bmod p$$
  - Calcola
  $$k_2 = H(z)$$
  Se $\gcd(k_2, p-1) \neq 1$ allora il dispositivo sceglie casualmente k_2 (l'attacco SETUP non è andato a buon fine).
  - Firma il messaggio M_2, dunque calcola
  $$r_2 \equiv g^{k_2} \bmod p \quad \text{e} \quad s_2 \equiv k_2^{-1}(M_2 - xr_2) \bmod p - 1$$
  - La coppia (r_2, s_2) è inviata a Bob per verificare l'autenticità del messaggio.

A questo punto, Eve non deve fare altro che collezionare le coppie (r_i, s_i). Dopodiché, è in grado di recuperare la chiave segreta x nel seguente modo.

- Eve calcola
$$R \equiv r_1^a g^b \bmod p$$

- Usando la sua chiave privata X calcola
$$z_1 \equiv \frac{r_1}{R^X} \bmod p$$

- A questo punto ci sono due possibilità.

  - Se $r_2 \equiv g^{H(z_1)}$, allora $H(z_1) = k_2$.
  - Altrimenti, Eve calcola
  $$z_2 \equiv \frac{z_1}{g^W}$$
  e stavolta $r_2 \equiv g^{H(z_2)} \bmod p$. Allora $H(z_2) = k_2$.

- In entrambi i casi, Eve riesce a recuperare k_2, a questo punto, non deve fare altro che calcolare
$$s_2 k_2 \equiv M_2 - xr_2 \bmod p - 1$$
ed estrapolare x dalla congruenza.

Notiamo che l'ultimo punto dell'algoritmo non è così immediato come sembra. Il problema è che la chiave privata x viene estrapolata da una congruenza modulo un numero pari $p-1$ e questo fatto non assicura in alcun modo l'invertibilità di r_2.

Riscriviamo la congruenza $s_2k_2 \equiv M_2 - xr_2 \bmod p-1$ nella forma più snella

$$ax \equiv b \bmod p-1 \tag{3.3}$$

dove $a = r_2$ e $b = M_2 - s_2k_2$. Se $\gcd(a, p-1) = 1$ allora è possibile calcolarsi $a^{-1} \bmod p-1$ ed estrapolare direttamente x.

Se $\gcd(a, p-1) = d \neq 1$, bisogna prima di tutto verificare che il sistema ammetta soluzioni. Ciò accade se e solo se $d \,|\, b$. Ma $b = M_2 - s_2k_2 \equiv M_2 - (M_2 - xr_2) \equiv xr_2 \bmod p-1$ dunque d divide b (modulo $p-1$). Chiarito questo fatto, non resta che andare in cerca delle soluzioni. Trovata una soluzione particolare x_0 della congruenza (3.3), possiamo elencare tutte le soluzioni, le quali hanno la seguente forma:

$$x_i = x_0 + i\Big(\frac{p-1}{d}\Big) \quad i = 0, 1, 2, \ldots, d-1 \tag{3.4}$$

Come si trova x_0? Esso è soluzione del sistema ridotto

$$\frac{a}{d}x \equiv \frac{b}{d} \bmod \frac{p-1}{d}$$

Infatti, $\gcd(a/d, (p-1)/d) = 1$, da cui

$$x \equiv \Big(\frac{b}{d}\Big)\Big(\frac{a}{d}\Big)^{-1} \bmod \frac{p-1}{d}$$

Dunque, se d è relativamente piccolo, il calcolo di x non è un problema per Eve.

Vediamo un esempio concreto del funzionamento di questo attacco.

Esempio 14. • Per prima cosa, Eve genera la sua coppia di chiavi. Per farlo ha bisogno di p e di g. Supponiamo che $p = 151$ e $g = 6$. A questo punto, Eve sceglie $X = 51$ e calcola la sua chiave pubblica

$$Y \equiv g^X \equiv 41 \bmod 151$$

- Alice vuole inviare dei messaggi a Bob e firmarli mediante lo schema di ElGamal. Prima di tutto sceglie la sua chiave privata. Supponiamo $x = 18$.
- A questo punto, Alice vuole inviare il messaggio $M_1 =$"SONO" a Bob. Il primo passo è la codifica di M_1 tramite un numero intero, supponiamo $M_1 = 147$. Quindi, Alice sceglie

  $$k_1 = 121$$

  e consta che $\gcd(121, 150) = 1$. Adesso può firmare il messaggio calcolando

  $$r_1 \equiv g^{k_1} \equiv 82 \bmod 151$$

  ed

  $$s_1 \equiv k_1^{-1}(M_1 - xr_1) \equiv 31 \cdot (147 - 18 \cdot 82) \equiv 51 \bmod 150$$

  Quindi invia la coppia (r_1, s_1) e il messaggio a Bob. Quest'ultimo seguirà la classica routine per autenticare il messaggio.

- Per firmare il secondo messaggio M_2 ="IO", codificato come $M_2 = 78$, Alice deve calcolare una nuova coppia (r_2, s_2). Tuttavia, Eve si era preventivamente impossessata del dispositivo di Alice, in cui ha installato Y e gli interi $a = 1, b = 4, W = 111$. Supponiamo inoltre che $t = 0$. Quindi, Alice chiede di generare la chiave segreta, ma il dispositivo calcola

$$\begin{aligned} z &\equiv g^{k_1 - Wt} Y^{-ak_1 - b} \\ &\equiv 6^{121 - 111 \cdot 0} 41^{-1 \cdot 121 - 4} \\ &\equiv 6^{121} 41^{-125} \\ &\equiv 69 \bmod 151 \end{aligned}$$

  A questo punto, con la funzione H, il dispositivo produce la seconda chiave

$$k_2 = H(z) = H(69) = 29$$

  il quale è coprimo con 150. A questo punto, Alice calcola

$$r_2 \equiv g^{k_2} \equiv 35 \bmod 151$$

  e

$$s_2 \equiv k_2^{-1}(M_2 - xr_2) \equiv 119 \cdot (78 - 18 \cdot 35) \equiv 12 \bmod 150$$

  Il tutto è inviato a Bob .

- Ora Eve può portare a termine il suo attacco. Colleziona (r_1, s_1) ed (r_2, s_2), poi calcola

$$R \equiv r_1^a g^b \equiv 82^1 6^4 \equiv 119 \bmod 151$$

  Successivamente calcola

$$z_1 \equiv \frac{r_1}{R^X} \equiv \frac{82}{119^{51}} \equiv 69 \bmod 151$$

  Ora, Eve calcola

$$g^{H(z_1)} \equiv 6^{H(69)} \equiv 35 \bmod 151$$

  e nota che coincide con r_2. Quindi conclude che $z_1 = z$, ma sopratutto

$$k_2 = H(z) = H(z_1) = 29$$

  Infine, usando k_2, Eve può calcolare x da s_2.

$$s_2 k_2 \equiv 48 \bmod 150$$

  ma $s_2 k_2 \equiv M_2 - xr_2 \bmod p - 1$. Quindi $xr_2 \equiv 78 - 48 \equiv 30 \bmod 150$. Siccome r_2 non è coprimo con 150 non è possibile calcolarsi direttamente x. Tuttavia, $\gcd(r_2, p - 1) = 5$. Quindi, esistono al più 5 soluzioni modulo 150 e sono tutte del tipo

$$x_i = x_0 + i(30) \quad i = 0, 1, 2, 3, 4$$

dove x_0 è una soluzione particolare. Tale x_0 è soluzione del sistema ridotto

$$x(35/5) \equiv (30/5) \bmod (150/5) \quad \text{cioè} \quad 7x \equiv 6 \bmod 30$$

Ora, $\gcd(7, 30) = 1$ quindi

$$x \equiv 6 \cdot 7^{-1} \equiv 18 \bmod 30$$

Dunque $x_0 = 18$ e coincide con la chiave privata di Alice, se non fosse stato così, allora bastava applicare la formula (3.4) e verificare la congruenza.

Adesso Eve dispone della chiave privata di Alice x.

Coprimalità di k_2 ed $p - 1$

Quello di r_2 non è il solo problema dell'invertibilità, anche k_2 deve soddisfare la condizione

$$\gcd(k_2, p - 1) = 1 \tag{3.5}$$

Se così non fosse, il dispositivo ricalcola k_2 in modo casuale ed Eve perderebbe la possibilità di estrapolare la chiave privata di Alice.

Assumiamo che Eve abbia collezionato due firme consecutive qualsiasi dette A_i ed A_{i+1}. La probabilità che queste due firme siano utili per estrapolare la chiave privata x dipende dalla probabilità che l'equazione (3.5) sia verificata. Per calcolare tale probabilità, chiediamoci prima di tutto quale è la probabilità che due numeri interi casuali siano coprimi tra loro. Vale il seguente Teorema, di cui alcune dimostrazioni si possono trovare in [1].

Teorema 8. *Siano a e b due numeri interi positivi scelti a caso. Allora*

$$P(\gcd(a, b) = 1) = \frac{6}{\pi^2}$$

Quindi, assumiamo che k_2 e $p - 1$ siano scelti a caso, allora

$$P(\gcd(k_2, p - 1) = 1) = \frac{6}{\pi^2} \approx 61\%$$

Tuttavia, k_2 non è casuale ma dipendente da parametri. Nonostante ciò, vedremo nella sezione relativa alla sicurezza, che $H(z)$ è uniformemente distribuito in $\mathbb{Z}_p$, e quindi è ragionevole supporre k_2 come casualmente scelto.

Inoltre, l'algoritmo funziona a coppie, nel senso che, una volta generate (r_1, s_1) ed (r_2, s_2) esse sono fine a se stesse, indipendenti dalle successive (r_3, s_3), (r_4, s_4). Quindi, per aumentare la probabilità che due firme consecutive qualunque $A_i = (r_1, s_i)$ ed $A_{i+1} = (r_{i+1}, s_{i+1})$ siano efficaci per estrapolare la chiave di Alice, possiamo far dipendere le firme successive $A_{i+2}, A_{i+3}, \ldots$ dalle precedenti. Dunque, una generica firma A_{i+1} dipenderà sempre da A_i.

Per realizzare questo programma, si potrebbe inserire nell'algoritmo sopra un loop, in modo tale da rigenerare la firma A_i fintato che non viene soddisfatta la condizione

$$\gcd(k_{i+1}, p-1) = 1$$

Uno schizzo del programma potrebbe essere il seguente.

- Si fissa inizialmente il contatore $j = 0$.
- Il dispositivo sceglie casualmente $k_i \in \{2, \ldots, p-2\}$ tale che $\gcd(k_i, p-1) = 1$.
- Dopodiché, viene calcolato k_{i+1} nel seguente modo.

  $$z \equiv g^{k_i - Wt} \cdot Y^{-ak_i - b} \bmod p$$

  e

  $$k_{i+1} = H(z)$$

  il quale deve soddisfare la condizione

  $$\gcd(k_{i+1}, p-1) = 1 \tag{3.6}$$

- A questo punto ci sono tre possibilità:

  1. Se la (3.6) non è verificata, allora $j = j + 1$ e il programma genera un nuovo k_i. Il processo viene ripetuto al più B volte.
  2. Se la (3.6) non è verificata e $j = B$, allora k_{i+1} è generato casualmente in $\{1, \ldots, p-2\}$.
  3. Se la (3.6) è verificata, allora il dispositivo calcola le firme (r_i, s_i) ed (r_{i+1}, s_{i+1}) secondo il protocollo standard del SETUP.

Quindi viene aggiunto un ciclo con condizione di uscita $j = B$. Adesso, bisogna capire che valore possiamo attribuire a B affinché la probabilità di ottenere due firme consecutive utili sia elevata. Dunque, vogliamo massimizzare

$$P(\gcd(k_{i+1}, p-1) = 1 \text{ in al più } B \text{ iterazioni}) = 1 - P(\gcd(k_{i+1}, p-1) \neq 1 \text{ dopo } B \text{ iterazioni})$$

ma

$$P(\gcd(k_{i+1}, p-1) \neq 1 \text{ dopo } B \text{ iterazioni}) = \left(1 - \frac{6}{\pi^2}\right)^B$$

quindi

$$P(\gcd(k_{i+1}, p-1) = 1 \text{ in al più } B \text{ iterazioni}) = 1 - \left(1 - \frac{6}{\pi^2}\right)^B$$

Già per $B = 7$ abbiamo che

$$P(\gcd(k_{i+1}, p-1) = 1 \text{ in al più } 7 \text{ iterazioni}) \approx 99\%$$

Va notato che questa è una stima ottimistica poiché il teorema 8 asserisce che gli interi possono essere qualsiasi. In particolare, una volta fissato $p-1$ esistono una infinità di numeri coprimi con esso. Nella pratica, sappiamo che k_{i+1} è un valore minore di $p-1$,

quindi ci sono meno interi che potrebbero essere coprimi con $p-1$ e, di conseguenza, anche la relativa probabilità $P(\gcd(k_{i+1}, p-1) = 1)$ sarà più bassa. In effetti essa vale:

$$P(\gcd(k_{i+1}, p-1) = 1) = \frac{\phi(p-1)}{p-1}$$

dove ϕ è la funzione di Eulero.

Simulando il discorso già fatto per il SETUP in DH, questa probabilità è alta se usiamo un primo sicuro $p = 2q + 1$, per cui

$$P(\gcd(k_{i+1}, p-1) = 1) \approx \frac{1}{2}$$

Con questa stima, otteniamo che

$$P(\gcd(k_{i+1}, p-1) = 1 \text{ in al più 7 iterazioni}) \approx 99\%$$

Riassumendo, è possibile modificare l'originale algoritmo SETUP aggiungendo un loop che garantisce la dipendenza delle firme $A_{i+1}, A_{i+2}, A_{i+3}, \ldots$, rispettivamente da $A_i, A_{i+1}, A_{i+2}, \ldots$. Supponendo che Eve riesca a collezionare almeno 7 firme consecutive (quindi 6 coppie papabili), la probabilità che almeno una coppia di firme consecutive sia utilizzabile per calcolare la chiave privata di Alice è estremamente elevata, tanto da rasentare la certezza.

Sicurezza

Discuteremo brevemente la sicurezza per questo schema; essa ripercorre esattamente gli stessi passi seguiti in 3.3.

Supponendo che un utente non conosca k_1, sarebbe improponibile ricavarsi k_2 poiché la difficoltà del problema è equivalente a quel di un DHP (Teorema 6).

Inoltre, sotto opportune ipotesi e grazie alla casualità insita in k_1, l'elemento di campo nascosto z appare uniformemente distribuito in $\mathbb{Z}_p^\times$ (teorema 7). Con questo risultato e supponendo la pseudocasualità di H, si può dimostrare che gli output di C e C' sono polinomialmente indistinguibili (corollario 1). Inoltre, un utente che non conosce la chiave privata X non può risalire efficacemente a z, e quindi a k_2.

Nonostante tutte questi risultati, il SETUP implementato nel protocollo di firma digitale di ElGamal è un SETUP regolare. In effetti, supponendo che Alice possa analizzare il dispositivo prima e dopo ogni suo utilizzo, allora può conoscere la sua chiave privata x. Quindi, dalla definizione di $s_2 \equiv k_2^{-1}(M_2 - xr_2) \bmod p-1$, la ragazza può risalire al valore k_2 (essa conosce l'effettiva implementazione del SETUP) e analogamente a quello di k_1. Con questo metodo, può risalire a tutti gli esponenti precedentemente generati. Se, inoltre, conosce a, b e il seme di H, allora Alice può definitivamente rilevare la presenza di un meccanismo SETUP nel suo dispositivo come già detto nella sezione 3.3. Questo va in contrasto con il punto 7 della definizione 3.5. Quindi il SETUP è regolare.

3.3.4 SETUP in DSA

Anche per l'algoritmo DSA ci sono due possibilità di implementazione di un DLK. La prima è sita nella generazione delle chiavi, ma essendo questa del tutto analoga a quella

di ElGamal, il risultato è un SETUP inefficiente. Per questo motivo, ci limiteremo a descrivere il meccanismo SETUP volto a manomettere il processo di firma, sfruttando l'elemento k.

Ricordiamo che Alice genera casualmente la sua chiave privata $x \in \{1, \dots, q-1\}$ dove q è numero primo, divisore di p. Inoltre, la firma sfrutta una funzione di hash h per prevenire attacchi di falsificazione digitale.

Algoritmo

Supponiamo che Alice voglia mandare un paio di messaggi M_1, M_2 a Bob e firmarli, utilizzando il protocollo DSA. Per far ciò, Alice acquisisce la chiave pubblica di Bob (p, q, g, k_B) dove p, q e g sono parametri fissi. Eve è riuscita ad impadronirsi del dispositivo di Alice e lo ha manomesso installando un meccanismo SETUP che consta dei seguenti dati: la chiave pubblica di Eve Y, tre interi fissati a, b, W, con quest'ultimo dispari, e una funzione di hash H crittograficamente sicura, che genera valori più piccoli di $q-1$.

Vediamo come si comporta il dispositivo contaminato di Alice.

- La prima volta che il dispositivo viene usato, cioè Alice manda il primo messaggio M_1 a Bob, esegue:
  - Genera la coppia di chiavi x, $g^x \bmod p$ di Alice.
  - Sceglie uniformemente a caso un intero $k_1 \in \{1, \dots, q-1\}$
  - Per firmare il messaggio M_1, il dispositivo calcola

$$r_1 \equiv (g^{k_1} \bmod p) \bmod q \quad \text{e} \quad s_1 \equiv k_1^{-1}(h(M_1) + xr_1) \bmod q$$

  - La coppia (r_1, S_1) e il messaggio M_1 sono inviati a Bob, il quale verifica l'autenticità della firma. Inoltre, k_1 è salvata in una memoria non volatile.
- La seconda volta che il dispositivo viene usato, cioè per mandare un secondo messaggio M_2, esso esegue:
  - Sceglie uniformemente a caso un intero $t \in \{0, 1\}$.
  - Calcola

$$z \equiv g^{k_1 - Wt} \cdot Y^{-ak_1 - b} \bmod p$$

  - Calcola

$$k_2 = H(z)$$

  - Firma il messaggio M_2, dunque calcola

$$r_2 \equiv (g^{k_2} \bmod p) \bmod q \quad \text{e} \quad s_2 \equiv k_2^{-1}(h(M_2) + xr_2) \bmod q$$

  - La coppia (r_2, S_2) è inviata a Bob per verificare l'autenticità del messaggio.

A questo punto, Eve non deve fare altro che collezionare le coppie (r_i, S_i). Dopodiché, è in grado di recuperare la chiave segreta x nel seguente modo.

- Calcola
$$R \equiv (g^{k_1})^a g^b \bmod p$$
- Usando la sua chiave privata X calcola
$$z_1 \equiv \frac{g^{k_1}}{R^X} \bmod p$$
- A questo punto ci sono due possibilità.
  - Se $r_2 \equiv g^{H(z_1)} \bmod q$, allora $H(z_1) = k_2$.
  - Altrimenti, Eve calcola
$$z_2 \equiv \frac{z_1}{g^W} \bmod p$$
    e stavolta $r_2 \equiv g^{H(z_2)} \bmod q$. Allora $H(z_2) = k_2$.
- In entrambi i casi, Eve riesce a recuperare k_2, a questo punto, non deve fare altro che calcolare
$$s_2 k_2 \equiv h(M_2) + x r_2 \bmod q$$
  ed estrapolare x dalla congruenza.

Notiamo che per estrapolare il cleptogramma r_2 Eve necessità del valore $g^{k_1} \bmod p$. Questo è facilmente calcolabile dalla verifica della prima firma (2.4.4). Infatti

$$g^{s_1^{-1} h(M_1)} \cdot k_A^{s_1^{-1} \cdot r_1} \bmod p \equiv (g^{k_1} \bmod p) \bmod q$$

dove k_A è la chiave pubblica di Alice. Quindi

$$g^{k_1} \bmod p = g^{s_1^{-1} h(M_1)} \cdot k_A^{s_1^{-1} \cdot r_1} \bmod p$$

poiché g è un elemento di ordine q modulo p.

Per il resto, la correttezza di questo SETUP è del tutto analoga a quella già descritta per il DH.

Osservazione 6. Nonostante l'analogia con lo schema di firma digitale di ElGamal, stavolta i calcoli sono fatti modulo q, il quale è un numero primo. Per tali motivo scompaiono i problemi legati all'invertibilità di r_2 ed k_2. Con le stesse considerazioni, notiamo che un attacco SETUP implementato nel DSA si rileva di gran lunga più efficace di quello in ElGamal, in quanto una coppia consecutiva di firme sarà sempre utile per portare a termine l'attacco.

Esempio 15. • Per prima cosa, Eve genera la sua coppia di chiavi. Per farlo ha bisogno di p,q e di g. Supponiamo che $p = 179$ e $q = 89$ e $g = 3$. A questo punto, Eve sceglie $X = 51$ e calcola la sua chiave pubblica
$$Y \equiv g^X \equiv 68 \bmod 179$$

- Alice vuole inviare dei messaggi a Bob e firmarli mediante lo schema DSA. Prima di tutto sceglie la sua chiave privata. Supponiamo $x = 18$.

- A questo punto, Alice vuole firmare il messaggio M_1 ="SONO". Il primo passo è la codifica di M_1 tramite la funzionde di hash h, supponiamo $h(M_1) = 32$. Quindi, Alice sceglie
$$k_1 = 7$$
Adesso può firmare il messaggio calcolando
$$r_1 \equiv g^{k_1} \bmod p \equiv 3^7 \bmod 179 \equiv 39 \bmod 89$$
ed
$$s_1 \equiv k_1^{-1}(h(M_1) + xr_1) \equiv 51 \cdot (32 + 18 \cdot 39) \equiv 54 \bmod 89$$
Quindi invia la coppia (r_1, s_1) e il messaggio a Bob. Quest'ultimo seguirà la classica routine per autenticare il messaggio.

- Per firmare il secondo messaggio M_2 ="IO", codificato come $h(M_2) = 78$, Alice deve calcolare una nuova coppia (r_2, s_2). Tuttavia, Eve si era preventivamente impossessata del dispositivo di Alice, in cui ha installato Y e gli interi $a = 1, b = 4, W = 111$. Supponiamo inoltre che $t = 0$. Quindi, Alice chiede di generare la chiave segreta, ma il dispositivo calcola
$$\begin{aligned} z &\equiv g^{k_1 - Wt} Y^{-ak_1 - b} \\ &\equiv 3^{7-111\cdot 0} 38^{-1\cdot 7 - 4} \\ &\equiv 3^7 68^{-11} \\ &\equiv 110 \bmod 179 \end{aligned}$$
A questo punto, con la funzione H, il dispositivo produce la seconda chiave
$$k_2 = H(z) = H(110) = 29$$
Ora, Alice calcola
$$r_2 \equiv g^{k_2} \bmod p \equiv 3^{29} \bmod 179 \equiv 46 \bmod 89$$
e
$$s_2 \equiv k_2^{-1}(h(M_2) + xr_2) \equiv 43 \cdot (78 + 18 \cdot 46) \equiv 65 \bmod 89$$
Il tutto è inviato a Bob .

- Ora Eve può portare a termine il suo attacco. Colleziona (r_1, s_1) ed (r_2, s_2), poi calcola
$$R \equiv (g^{k_1})^a g^b \equiv 3^7 3^4 \equiv 116 \bmod 179$$
Successivamente calcola
$$z_1 \equiv \frac{g^{k_1}}{R^X} \equiv \frac{3^7}{116^{51}} \equiv 110 \bmod 179$$

Ora, Eve calcola

$$g^{H(z_1)} \bmod p \equiv 3^{H(110)} \bmod 179 \equiv 46 \bmod 89$$

e nota che coincide con r_2. Quindi conclude che $z_1 = z$, ma sopratutto

$$k_2 = H(z) = H(z_1) = 29$$

Infine, usando k_2, Eve può calcolare x da s_2.

$$x \equiv r_2^{-1}(s_2 k_2 - h(M_2)) \equiv 60 \cdot (65 \cdot 29 - 78) \equiv 18 \bmod 89$$

Adesso Eve dispone della chiave privata di Alice x.

Sicurezza

Anche per il DSA vale un discorso analogo a quanto fatto per i sitemi di ElGamal: la trattazione generale della sicurezza ricalca quella dello scambio di chiavi di Diffie-Hellamn.

Supponendo che un utente non conosca k_1, sarebbe improponibile ricavarsi k_2 poiché la difficoltà del problema è equivalente a quel di un DHP (Teorema 6).

Inoltre, sotto opportune ipotesi e grazie alla casualità insita in k_1, l'elemento di campo nascosto z appare uniformemente distribuito in $\mathbb{Z}_p^\times$ (teorema 7). Con questo risultato e supponendo la pseudocasualità di H, si può dimostrare che gli output di C e C' sono polinomialmente indistinguibili (corollario 1). Inoltre, un utente che non conosce la chiave privata X non può risalire efficacemente a z, e quindi a k_2.

Nonostante tutte questi risultati, il SETUP implementato nel protocollo di firma digitale DSA è un SETUP regolare. In effetti, supponendo che Alice possa analizzare il dispositivo prima e dopo ogni suo utilizzo, allora può conoscere le sue chiavi private x. Quindi, dalla definizione di $s_2 \equiv k_2^{-1}(h(M_2) + x r_2) \bmod q$, la ragazza può risalire al valore k_2 (essa conosce l'effettiva implementazione del SETUP) e analogamente per k_1. Con questo metodo, può risalire a tutti gli esponenti precedentemente generati. Se, inoltre, conosce a, b e il seme di H, allora Alice può definitivamente rilevare la presenza di un meccanismo SETUP nel suo dispositivo come già detto nella sezione 3.3. Questo va in contrasto con il punto 7 della definizione 3.5. Quindi il SETUP è regolare.

DSA e canali subliminali

L'implementazione del SETUP nel DSA può essere utile per avviare un canale subliminale tra Alice e Bob.

La forza di questo canale è che Alice può far trapelare la sua chiave privata a Bob senza che i due si incontrino preventivamente.

Dunque, supponiamo che Alice sia stata arrestata e voglia comunicare con Bob (all'esterno) per pianificare una fuga. Purtroppo non sono riusciti a scambiarsi le chiavi private a vicenda prima della cattura. Tuttavia, entrambi sanno come funziona il SETUP su DSA e quindi Alice implementa tale meccanismo nel suo dispositivo.

Quindi, supponiamo che p sia un primo di 1024 bit e q di 160 bit. Supponiamo che Alice voglia mandare un messaggio $M < q$ di 160 bit.

A tal scopo:

- Alice genera due messaggi innocui M_1 e M_2.
- A questo punto, serve un generatore g_1 di $\mathbb{Z}_q^\times$ in $\mathbb{Z}_p$. Tale elemento dipenderà da M_1.

  Quindi, detto $w > 0$ il più piccolo valore tale che $h(M_1)^w \bmod p$ genera $\mathbb{Z}_p^\times$, Alice pone
  $$g_1 \equiv (h(M_1)^w)^{(p-1)/q} \bmod p$$
  Dunque, g_1 è un elemento di ordine q in $\mathbb{Z}_p$.
- Usando la sua chiave privata x, Alice calcola $k_1 = ((g_1^x \bmod p) \bmod q) \cdot M \bmod q$ da cui
  $$r_1 \equiv (g^{k_1} \bmod p) \bmod q \quad \text{e} \quad s_1 \equiv k_1^{-1}(h(M_1) + xr_1) \bmod q$$
  e la coppia (r_1, s_1) costituisce la firma al primo messaggio M_1.
- Ora entra in scena il SETUP che calcola k_2 nel consueto modo tramite l'elemento di campo nascosto z. Quindi, Alice calcola la firma per il secondo messaggio
  $$r_2 \equiv (g^{k_2} \bmod p) \bmod q \quad \text{e} \quad s_2 \equiv k_2^{-1}(h(M_2) + xr_2) \bmod q$$

A questo punto, i due messaggi insieme alle relative firme sono inviate a Bob, il quale può recuperare M come segue:

- Bob, collezionando (r_2, s_2) può ricavare la chiave privata x di Alice nel consueto modo (calcola R, z_1, ecc...).
- Con la chiave privata x, Bob può calcolarsi
  $$k_1 \equiv s_1^{-1}(h(M_1) + xr_1) \bmod q$$
- Bob conosce M_1 e quindi può calcolarsi g_1 esattamente come ha fatto Alice.
- Infine, Bob calcola
  $$M \equiv k_1((g_1^x \bmod p) \bmod q)^{-1} \bmod q$$

Quindi, con questo algoritmo, Alice manda il messaggio celato M ed un cleptogramma nella prima firma. Il cleptogramma è quindi utilizzato per manomettere la seconda firma. Secondo lo schema SETUP, Bob può ricavare k_2 da cui scopre x dalla seconda firma. Usando x e la prima firma, Bob recupera M.

Notiamo che tale sistema può essere utilizzato una sola volta; infatti, k_1 non è generato casualmente secondo il desueto schema del DSA, ma dipende fortemente da M_1. Quindi, se Alice usasse gli stessi messaggi M, M_1 e M_2 per comunicare anche con Carol, la guardia Eve potrebbe accorgersi della raggiro e tagliare le comunicazioni. Per tale motivo è meglio usare parte di quei 160 bit di M per introdurre casualità nel sistema.

Eve è furba, e sospetta che Alice e Bob la stiano ingannando comunque. Tuttavia, vuole essere certa dell'imbroglio e decide di indovinare M. Ricordiamo che ciò non è una idea grottesca, in quanto M è solo di 160 bit, quindi Eve può utilizzare la forza bruta. Ora, se fosse $k_1 = M$, allora Eve potrebbe azzeccare M calcolando $(g^M \bmod p) \bmod q$ e confrontandolo con $r_1 \equiv (g^{k_1} \bmod p) \bmod q$. Ma in questo settaggio, oltre ad M, Eve dovrebbe indovinare anche $(g_1^x \bmod p) \bmod q$. Un problema non semplice.

Capitolo 4

SETUP su Curve Ellittiche

L'analisi fatta nel capitolo precedente può essere traslata e adattata anche al contesto delle curve ellittiche, tanto è vero che, gli stessi Young e Yung ne parlano in alcuni loro articoli [57, 51]. In particolare essi affermano che

> The discrete log setup extends directly to elliptic curve cryptosystems.

In questo capitolo, quindi descriveremo più succintamente i meccanismi del *Elliptic Curve Discrete Log Kleptogram* (ECDLK) implementati nello scambio di chiavi di Diffie-Hellman su curve ellittiche (ECDH), i protocolli di ElGamal (ECEE e ECES) e il DSA (ECDSA). Per far ciò, seguiremo gli articoli [27, 28, 29] e [38].

Le motivazioni principali che ci spingono a trattare questi argomenti sono le potenzialità che le curve ellittiche offrono dal punto di vista dell'efficienza e della sicurezza. Inoltre, nelle curve ellittiche, viene meno il concetto di residuo quadratico rendendo la discussione fatta in 3.3 fallace. In questo senso i SETUP su curve ellittiche forniscono una maggiore sicurezza dal punto di vista de Eve.

Per ognuno degli attacchi presentati, forniremo inoltre un algoritmo esplicito. Questo servirà per discutere l'efficienza.

4.1 Elliptic Curve Discrete Log Kleptogram

L'implementazione di un SETUP in protocolli basati sul problema del logaritmo discreto può essere ampliata all'analogo problema sulle curve ellittiche ECDLP. Per questo motivo presentiamo l'algoritmo SETUP *Elliptic Curve Discrete Log Kleptogram* nei sistemi basati sul ECDLP.

Il funzionamento del ECDLK è del tutto simile a quello già presentato in 3.3. Esso si sviluppa in almeno due iterazioni del dispositivo contaminato, inoltre consta delle solite quattro parti: identificazione del cleptogramma, che stavolta non sarà una esponeziazione ma una moltiplicazione di un punto per uno scalare; memorizzazione del relativo scalare; manomissione per mezzo dell'elemento di campo nascosto e infine il recupero delle informazioni private.

Anticipiamo fin da subito che le somiglianze con il caso discreto saranno notevoli, in particolare la trattazione della sicurezza avrà un andamento del tutto analogo a quella descritta in 3.3.

Algoritmo

Alice e Bob vogliono comunicare per mezzo di un protocollo crittografico basato sull'ECDLP. Inizialmente viene scelta una curva ellittica E definita su un campo finito $\mathbb{F}_q$. Sia $G \in E(\mathbb{F}_q)$ un punto di ordine n. Supponiamo che il dispositivo pubblichi solamente il valore $[c]G$ dove $c \in \{2, \ldots, n-2\}$.

Anche stavolta Eve è riuscita ad impadronirsi del dispositivo di Alice, inserendo i seguenti dati: tre interi fissati a, b e w (dispari), la sua chiave pubblica $Y = [x]G$ e una funzione hash H crittograficamente sicura, che genera valori più piccoli di $\phi(n)$. In concreto, la funzione H fornisce l'impronta dell'ascissa P_x di un punto P. Per non appesantire la notazione, scriveremo indifferentemente $H(P)$ o $H(P_x)$, intendendo che $H(P) = H(P_x)$.

Il dispositivo contaminato di Alice esegue le seguenti operazioni.

- La prima volta che il dispositivo viene usato, esso esegue:
  - Sceglie a caso e in modo uniforme un intero $c_1 \in \{2, \ldots, n-2\}$.
  - Calcola e pubblica
    $$Y_1 = [c_1]G$$
  - c_1 è salvato in una memoria non volatile (disco rigido) in modo da essere utilizzato al secondo utilizzo dell'apparato.
- La seconda volta che il dispositivo viene usato, esso esegue:
  - Sceglie a caso e in modo uniforme un intero $t \in \{0, 1\}$.
  - Calcola *l'elemento di campo nascosto*
    $$Z = [c_1 - w \cdot t]G + [-a \cdot c_1 - b]Y \tag{4.1}$$
  - Calcola la seconda chiave privata di Alice nel seguente modo
    $$c_2 = H(Z_x)$$
  - Calcola e pubblica
    $$Y_2 = [x_2]G$$

Esattamente come nel caso non ellittico, Eve non deve fare altro che collezionare le due informazioni pubbliche Y_1 e Y_2. Dopodiché, è in grado di recuperare c_2 nel seguente modo.

- Calcola
  $$R = [a]Y_1 + [b]G$$
- Usando la sua chiave privata x calcola
  $$Z_1 = Y_1 - [x]R$$

- A questo punto ci sono due possibilità.
  - Se $Y_2 = [H(Z_1)]G$, allora $H(Z_1) = c_2$ e si conclude.
  - Altrimenti, Eve calcola
    $$Z_2 = Z_1 - [w]G$$
    e stavolta $Y_2 = [H(Z_2)]G$. Allora $H(Z_2) = c_2$ e si conclude.

Un eventuale intruso che voglia rubare c_2 dovrebbe conoscere la chiave privata x di Eve la quale non è disponibile nel dispositivo. In questo caso Y_2 funge da cleptogramma poiché è l'informazione pubblica celante i bit nascosti per il recupero di informazioni private. Infine, Z è l'elemento di campo nascosto.

L'algoritmo proposto è corretto; infatti, lavorando con Z_1 e ricordando che $Y = [x]G$ si ha

$$\begin{aligned} Z_1 &= Y_1 - [x]R \\ &= [c_1]G - [x]([a]Y_1 + [b]G) \\ &= [c_1]G - [x][ac_1 + b]G \\ &= [c_1]G + [-ac_1 - b]Y \end{aligned} \tag{4.2}$$

Notiamo che tale valore coincide con (4.1) per $t = 0$. Quindi $Z_1 = Z$, da cui $c_2 = H(Z) = H(Z_1)$.

Se, invece, il dispositivo lavora con $t = 1$, si ha

$$\begin{aligned} Z_2 &= Z_1 - [w]G \\ &= [c_1]G + [-ac_1 - b]Y - [w]G \\ &= [c_1 - w]G + [-ac_1 - b]Y \end{aligned} \tag{4.3}$$

Anche qui ritroviamo la (4.1) per $t = 1$. Quindi $Z_2 = Z$ e $c_2 = H(Z) = H(Z_2)$.

Osservazione 7. Ricordiamo che, nell'implementazione originale proposta da Young e Yung, l'intero w serve per precauzione nel caso un utente riesca a scorgere qualche relazione probabilistica basata su certe proprietà di Y e di Z. Nel caso discreto di $\mathbb{Z}_p^\times$, tale connessione era strettamente legata alla presenza dei residui quadratici. Tuttavia, nelle curve ellittiche, non ha senso parlare di residui quadratici, per cui potrebbe essere un lavoro significativamente arduo quello di trovare un qualche legame probabilistico tra i suddetti valori. Inoltre, il gruppo con cui si lavora nel caso originale è $\mathbb{Z}_p^\times$ contenente un numero pari di elementi di cui l'esatta metà è formata da residui quadratici e l'altra da non residui. Nelle, applicazioni concrete con le curve ellittiche invece, si tendono ad usare curve il cui gruppo $E(\mathbb{F}_q)$ ha cardinalità un numero primo dispari, rendendo ancora più complessa la rilevazione di legami probabilistici. In conclusione, l'utilizzo contemporaneo di w e della funzione H, in questa versione ellittica, potrebbe essere superfluo.

Tuttavia, il ruolo di a e b è ancora efficace negli algoritmi di firma digitale.

Sicurezza

Come si è potuto apprezzare leggendo le righe sopra, l'attacco ECDLK è del tutto simile al DLK descritto in 3.3. Per tale motivo, non deve sorprendere che il livello di sicurezza fornito da questa versione ellittica sarà analogo alla versione classica. Per completezza, riportiamo quindi la verifica della sicurezza.

Ricordiamo quali sono i principali problemi da risolvere

1. Il meccanismo deve garantire che solo ed esclusivamente Eve può ricavare c_2, chiudendo le porte ad altri malintenzionati.

2. Nessuno, eccetto Eve, deve capire che un attacco SETUP è in esecuzione.

Per il primo problema, il calcolo di c_2 deve essere proibitivo a qualunque utente esterno, anche se a conoscenza dei parametri interni Y, a, b, w. A tale scopo dimostriamo il seguente teorema.

Teorema 9. *L'attacco Elliptic Curve Discrete Log Kleptogram è sicuro se e solo se il ECDHP 2.8 è sicuro.*

Dimostrazione. $\Rightarrow$ Supponiamo esista una macchina oracolo A in grado di rompere l'attacco ECDLK. Questo vuol dire che, conoscendo Y e Y_1, l'oracolo calcola Z_1 per $t = 0$ e Z_2 per $t = 1$, da cui potrà ricavare la chiave c_2. Denotiamo questo fatto con

$$A(Y, Y_1) = (Z_1, Z_2)$$

Ciò che dobbiamo dimostrare è che è possibile conoscere $[uv]G$, dati $[u]G$ e $[v]G$ (ECDHP).

Ora, ricordiamo che $Z_1 = [c_1]G + [-ac_1 - b]Y$, $Z_2 = [c_1 - w]G + [-ac_1 - b]Y$ e che $Y_1 = [c_1]G$. Quindi,

$$A(Y, Y_1) = A(Y, [c_1]G) = ([c_1]G + [-ac_1 - b]Y, [c_1 - w]G + [-ac_1 - b]Y)$$

Ora, inseriamo $[u]G$ ed $[v]G$ nell'oracolo. Così facendo otteniamo che

$$A([v]G, [u]G) = ([u]G + [-au - b][v]G, [u - w]G + [-au - b][v]G)$$

Notiamo che, definendo

$$F = ([u]G + [-b][v]G) - Z_1$$

allora,

$$\begin{aligned} F = ([u]G + [-b][v]G) - ([u]G + [-au - b][v]G) = \\ [-b][v]G - [-au - b][v]G = \\ [-b][v]G - [-b][v]G - [-au][v]G = [a][uv]G \end{aligned} \quad (4.4)$$

da cui

$$[uv]G = [a]^{-1}F$$

Quindi, con una macchina che è in grado di rompere l'ECDLK, possiamo anche risolvere un ECDHP.

$\Leftarrow$ Viceversa, supponiamo esista un oracolo B capace di risolvere il ECDHP, cioè, dati $[u]G$ e $[v]G$, la macchina B può calcolare $[uv]G$. In formule

$$B([u]G, [v]G) = [uv]G$$

Dobbiamo dimostrare che, dati Y, a, b, w, G, Y_1, H, l'oracolo B sia in grado di risalire a Z_1 e Z_2, e quindi a Z, così da potersi calcolare $c_2 = H(Z)$. Poiché B conosce Y_1, a, G, b, essa si calcola $[a]Y_1 + [b]G$ e inserisce tale valore dentro la macchina insieme ad Y. Quindi, B calcola

$$\begin{aligned} B([a]Y_1 + [b]G, Y) &= B([a][c_1]G + [b]G, Y) \\ &= B([ac_1 + b]G, Y) \\ &= [ac_1 + b]Y \end{aligned}$$

A questo punto non resta che definire

$$F = [c_1]G - B([a]AY1 + [b]G, Y) = [c_1]G + [-ac_1 - b]Y$$

Quindi, se il dispositivo contaminato ha lavorato con $t = 0$ allora $F = Z$. Altrimenti sarà $F - [w]G = Z$. Infine $c_2 = H(Z)$.

Ricapitolando, con una macchina in grado di risolvere il problema di Diffie-Hellman su curve ellittiche, possiamo determinare la chiave c_2 conoscendo i soli parametri del dispositivo.

□

Questo teorema dimostra che, anche conoscendo i parametri interni installati da Eve, un utente dovrebbe risolvere un problema computazionalmente equivalente ad un ECDHP per determinare c_2 e quindi violare l'attacco SETUP.

Per quanto riguarda la seconda questione, Alice, o un altro utente, non deve essere in grado di capire se un attacco SETUP ECDLK è in atto semplicemente guardando gli output prodotti. Il discorso è analogo a quello sulla sicurezza del DLK: Z e quindi $H(Z)$ devono essere uniformemente distribuiti. In questo modo si potrà far apparire c_2 come se fosse casualmente scelto in maniera uniforme e nessuno sarebbe in grado di dire se tale chiave sia stata manomessa o meno.

A tale scopo, possiamo ripercorrere i passi che ci hanno portato ad enunciare il Teorema 7; tuttavia, balza all'occhio una differenza sottile, ma importante: nel caso discreto, il gruppo in questione era $\mathbb{Z}_p^\times$ il quale è sì un gruppo ciclico, ma di ordine $p - 1$, che non è un numero primo. Nel contesto delle curve ellittiche, invece, lavoriamo con il gruppo generato da G, il quale ha ordine un numero primo, dunque, tutti gli elementi di $\langle G \rangle$, eccetto l'identità, sono generatori del gruppo. In conclusione, possiamo immediatamente affermare, senza ulteriore ipotesi, che vale il seguente teorema.

Teorema 10. *L'elemento di campo nascosto Z è uniformemente distribuito in $\langle G \rangle$.*

Dimostrazione. Da $Z = [c_1 - wt]G + [-ac_1 - b]Y$ segue che

$$\begin{aligned} Z &= [c_1 - wt]G + [-ac_1 - b][x]G \\ &= [c_1]G - [wt]G - [xb]G + [-axc_1]G \\ &= [-xb - wt]G + [c_1][1 - xa]G \\ &= [\alpha]G + [c_1]G_1 \end{aligned}$$

Dove $\alpha = -xb - wt$ e $G_1 = [1 - xa]G$. Per costruzione, G_1 è generatore del gruppo $\langle G \rangle$, per tanto

$$[\alpha]G = [u]G_1$$

per un qualche $u \in \{1, \ldots, n\}$. Da cui

$$Z = [u + c_1]G_1$$

Ma c_1 è stato casualmente e uniformemente scelto in $\{2, \ldots, n-2\}$, quindi anche il punto Z è uniformemente distribuito in $\langle G \rangle$. □

L'unico falla in questo sistema è la possibilità che $1 - xa \equiv 0 \bmod ord(G)$, in questo modo si avrebbe che $G_1 = \mathcal{O}$. Tuttavia, Eve conosce x ed a e può preventivamente sceglierli affinché questo scenario non si verifichi.

Corollario 2. *Supponiamo che H sia una funzione pseudocasuale e che le specifiche del dispositivo utilizzato siano pubblicamente consultabili. Allora gli output di C e C' sono polinomialmente indistinguibili.*

Dimostrazione. Dal teorema precedente sappiamo che Z è uniformemente distribuito nel gruppo generato da G. Poiché H è una funzione pseudocasuale, allora $c_2 = H(Z)$ è uniformemente distribuito in $\mathbb{F}_q$. Dunque, i valori $[c_2]G$ emessi da C e C' hanno distribuzioni di probabilità polinomialmente indistinguibili. □

Come nel caso discreto, anche questo attacco è uno schema $(1, 2)$ che può essere iterato nel seguente modo.

Supponiamo che Alice voglia comunicare anche con Dave. Per farlo genera una terza chiave c_3 la quale non è casuale, ma

$$c_3 = H(Z)$$

dove

$$Z = [c_2 - wt]G + [-ac_2 - b]Y$$

Svolgendo i passi per il recupero di c_2, Eve è in grado di calcolarsi c_3. Iterando il processo, possiamo concludere che l'ECDLK è uno schema $(m, m + 1)$.

Riassumendo quanto dimostrato in questa sezione, possiamo affermare che:

Teorema 11. *L'attacco ECDLK costituisce un SETUP forte ed è uno schema di dispersione $(m, m + 1)$.*

Non si deve commettere l'errore di pensare che tutti gli schemi che presentano l'implementazione di un SETUP simile al ECDLK ereditano da questo le relative proprietà di sicurezza. Il punto fondamentale è in quale parte del protocollo sottostante l'ECDLK viene installato e quali altri output vengono pubblicati. In particolare, anticipiamo già che gli schemi di firma costituiranno un SETUP regolare e non forte.

4.1.1 SETUP in ECDH

Il meccanismo SETUP che ora ci accingiamo a descrivere è pressoché identico al ECDLK, anche per quanto riguarda la questione sicurezza e il ruolo dei parametri a, b e w. Per completezza riportiamo lo schema ed un esempio. Un pseudocodice che ne descrive il comportamento è disponibile nell'appendice B.

Richiamiamo velocemente quanto già detto nel capitolo 2.5.2. Supponiamo che Alice e Bob vogliano comunicare tra di loro; per farlo hanno bisogno di una chiava segreta. Decidono di usare lo scambio di chiavi di Diffie-Hellman su curve ellittiche. Quindi, entrambi scelgono una curva sicura E definita su un campo finito $\mathbb{F}_q$. Da qui scelgono un punto G di ordine un primo grande n. A questo punto, generano le chiavi private $a, b \in \mathbb{F}_q$ e trasmettono le chiavi pubbliche $A = [a]G, B = [b]G$.

Supponiamo che il dispositivo di Alice sia stato contaminato da un meccanismo SETUP installato da Eve. Affinché quest'ultima possa estrapolare la chiave privata di Alice, ha bisogno di due chiavi pubbliche, quindi di due esecuzioni del ECDH. Le chiavi pubbliche conterranno delle informazioni celate atte a recuperare la chiave privata. Ricordiamo, infine, che il device di Alice è black-box.

Algoritmo

Come accennato, sono necessarie due esecuzioni del protocollo di scambio. Quindi supponiamo che Alice comunichi con Bob generando la chiave privata x_1, e con Carol con la chiave x_2. Anche stavolta Eve è riuscita ad impadronirsi del dispositivo di Alice, inserendo i seguenti dati: tre interi fissati a, b e w (dispari), la sua chiave pubblica $Y = [x]G$ e una funzione hash H crittograficamente sicura, che genera valori più piccoli di n.

A questo punto il dispositivo compie le seguenti operazioni.

- La prima volta che il dispositivo viene usato, esso esegue:
  - Sceglie a caso un intero $x_1 \in \mathbb{F}_q$.
  - Calcola e pubblica
    $$A_1 = [x_1]G$$
  - A_1 è salvata in una memoria non volatile (disco rigido) in modo da essere utilizzata al secondo utilizzo dell'apparato.
- La seconda volta che il dispositivo viene usato, esso esegue:
  - Sceglie uniformemente a caso un intero $t \in \{0, 1\}$.
  - Calcola
    $$Z = [x_1 - w \cdot t]G + [-a \cdot x_1 - b]Y \tag{4.5}$$
  - Calcola la seconda chiave privata di Alice nel seguente modo
    $$x_2 = H(Z)$$
    (ricordiamo che per Osservazione basterebbe inviare solo l'ascissa del punto z, ovvero z_x, da cui $x_2 = H(z_x)$).

– Calcola e pubblica

$$A_2 = [x_2]G$$

Esattamente come nel caso non ellittico, Eve non deve fare altro che collezionare le due chiavi pubbliche A_1 e A_2. Dopodiché, è in grado di recuperare x_2 nel seguente modo.

- Eve calcola

$$R = [a]A_1 + [b]G$$

- Usando la sua chiave privata x calcola

$$Z_1 = A_1 - [x]R$$

- A questo punto ci sono due possibilità.

  – Se $A_2 = [H(Z_1)]G$, allora $H(Z_1) = x_2$ e si conclude.

  – Altrimenti, Eve calcola

  $$Z_2 = Z_1 - [w]G$$

  e stavolta $A_2 = [H(Z_2)]G$. Allora $H(Z_2) = x_2$ e si conclude.

Dal cleptogramma A_2, Eve può risalire alla seconda chiave privata x_2 di Alice. Una volta estrapolata, Eve può trovare la chiave segreta usata da Alice e Carol per comunicare. Infatti, detta x_C la chiave privata di Carol, la nostra malintenzionata conosce $G, x_2, [x_2]G, [x_C]G$ e vuole calcolare $[x_2 \cdot x_C]G$. Tale problema è relativamente semplice.

Esempio 16.

Il primo passo è quello di concordare una curva ellittica, supponiamo $E : y^2 = x^3+2x+2$ definita sula campo $\mathbb{F}_{173}$. Notiamo che $\#E(\mathbb{F}_{173}) = 193$ il quale è un numero primo. Inoltre, come punto iniziale viene scelto $G = (2, 35)$ anch'esso di ordine $n = 193$.

Ora, Eve può generare la sua coppia di chiavi e sceglie $x = 51$ come chiave privata, da cui può calcolarsi la pubblica

$$Y = [x]G = (59, 54)$$

Alice vuole comunicare con Bob e Carol. Per comunicare con Bob genera la seguente coppia di chiavi

$$x_1 = 123 \quad \text{e} \quad A_1 = [x_1]G = (100, 75)$$

ed $A1$ è inviata a Bob che seguirà la solita routine per condividere la sua chiave pubblica.

Per comunicare con Carol, Alice genera una seconda coppia di chiavi. Tuttavia, Eve si era preventivamente impossessata del dispositivo della ragazza, in cui ha inserito $a = 1$, $b = 4$ $w = 7$. Supponiamo inoltre che $t = 1$. Quando Alice genera la chiave privata x_2, in realtà il dispositivo calcola

$$\begin{aligned} Z &= [x_1 - wt]G + [-ax_1 - b]Y \\ &= [123 - 7 \cdot 1]G + [-1 \cdot 123 - 4]Y \\ &= [116]G + [-127]Y \\ &= (96, 143) \end{aligned}$$

Di questo punto Z, viene calcolata l'impronta dell'ascissa $Z_x = 96$ tramite la funzione H. Quindi,

$$x_2 = H(Z) = H(Z_x) = H(96) = 26$$

Adesso Alice può calcolarsi la chiave pubblica seguendo l'algoritmo standard,

$$A2 = [x_2]G = (119, 44)$$

Infine, Alice manda $A2$ (oppure la sola ascissa $A2_x$) a Carol.

Adesso tocca a Carol calcolarsi la propria coppia di chiavi, che è

$$x_C = 8 \quad \text{e} \quad C = [x_C]G = (96, 143)$$

quindi, invia ad Alice C.

A questo punto, Alice e Carol dispongono della chiave segreta per comunicare,

$$K = [x_2]C = [x_C]A_2 = (11, 12)$$

Eve entra in scena: colleziona A_1, A_2 ed C e sfrutta i primi due per calcolarsi la chiave privata di Alice x_2. Prima di tutto calcola

$$R = [a]A_1 + [b]G = [1]A_1 + [4]G = (47, 130)$$

successivamente calcola

$$Z_1 = A_1 1 - [x]R = A_1 1 - [51]R = (11, 12)$$

Notando che $[(86, 52) = H(Z_1)G \neq A_2 = (119, 44)$, Eve conclude che deve calcolarsi Z_2. Quindi,

$$Z_2 = Z_1 - [w]G = Z_1 - [7]G = (96, 143)$$

e in effetti $Z_2 = Z$ (in realtà bastava solo che $(Z_2)_x = Z_x$).

Infine, Eve ottiene x_2 poiché

$$x_2 = H(Z) = H(Z_2) = 26$$

Conoscendo x_2 e A_C, Eve può calcolarsi la chiave segreta K. Infatti,

$$K = [x_2]C = (11, 12)$$

Nell'appendice B è possibile consultare i pseudocodici 3 ed 4 relativi all'implementazione del SETUP in ECDH.

Sicurezza

Il ruolo degli interi a, b, w serve come precauzione nel caso Alice sia in grado di invertire la funzione di hash H e riesca a scorgere qualche legame probabilistico tra gli output del dispositivo. Tali legami, come già detto nell'osservazione 7 sono difficili rilevare, rendendo il SETUP su ECDH in un certo senso più sicuro rispetto all'analogo non ellittico.

Per ciò che concerne la natura del SETUP presentato, possiamo seguire gli stessi passaggi logici descritti in 4.1 e concludere che

Teorema 12. *L'attacco SETUP implementato nello scambio di chiavi di Diffie-Hellman su curve ellittiche costituisce un SETUP forte ed è uno schema di dispersione* $(1, 2)$ *iterabile ad uno* $(m, m + 1)$.

4.1.2 SETUP in ECEE

Ricordiamo che nel sistema di cifratura di ElGamal su curve ellittiche gli utenti scelgono a priori la curva ellittica E ed il campo finito dove essa è definita, insieme al punto iniziale G di ordine n. Inoltre, generano anche la coppia di chiavi privata-pubblica (u, U) dove $u \in \{1, \ldots, n-1\}$ e $U = [u]G$. Notiamo che tali valori non cambieranno mai (salvo specifiche richieste da parte dell'utente o problemi tecnici). Per tale motivo, un ECDLK implementato durante la fase di generazione delle chiavi non sarebbe molto realistico, in quanto sono necessarie due chiavi per portare a termine l'attacco.

Se un utente vuole mandare un messaggio M, prima lo identifica con un punto di E ed invia la coppia (R, S) dove $R = [k]G$ per un qualche intero k casualmente scelto, ed $S = M + [k]U$.

Algoritmo

Supponiamo che Alice voglia mandare due messaggi M_1, M_2 a Bob utilizzando ECEE. Per far ciò, acquisisce la chiave pubblica di Bob (E, q, G, B) (la funzione f non la riportiamo poiché è quella che restituisce l'ascissa di un punto). Sia n l'ordine di G, il quale è un primo grande.

Eve è riuscita ad impadronirsi del dispositivo di Alice e lo ha manomesso installando un meccanismo SETUP che consta dei seguenti dati: la chiave pubblica di Eve Y, tre interi a, b, w e una funzione di hash H crittograficamente sicura che genera valori minori di $n - 1$.

- La prima volta che il dispositivo viene usato compie i seguenti passi:
  - Sceglie a caso un intero $k_1 \in \{1, \ldots, n-1\}$.
  - Cifra il messaggio M_1 secondo l'usuale schema ECEE, quindi calcola

$$R_1 = [k_1]G \quad \text{e} \quad S_1 = M_1 + [k_1]B$$

- La coppia (R_1, S_1) è inviata a Bob che la usa per decifrare il messaggio. Inoltre, k_1 è salvato in una memoria non volatile (disco rigido) in modo da essere utilizzato al secondo utilizzo dell'apparato.

- La seconda volta che il dispositivo viene usato compie i seguenti passi:

  - Sceglie uniformemente a caso un intero $t \in \{0, 1\}$.
  - Calcola
  $$Z = [k_1 - w \cdot t]G + [-a \cdot k_1 - b]Y \tag{4.6}$$
  - Calcola k_2 nel seguente modo
  $$k_2 = H(Z)$$
  - Cifra il messaggio M_2 secondo ECEE, quindi calcola
  $$R_2 = [k_2]G \quad \text{e} \quad S_2 = M_2 + [k_2]B$$

Esattamente come nel caso non ellittico, Eve non deve fare altro che collezionare le coppie (R_1, S_1) e (R_2, S_2). Dopodiché, è in grado di recuperare M_2 nel seguente modo.

- Eve calcola
$$R = [a]R_1 + [b]G$$
- Usando la sua chiave privata x calcola
$$Z_1 = R_1 - [x]R$$
- A questo punto ci sono due possibilità.

  - Se $R_2 = [H(Z_1)]G$, allora $H(Z_1) = k_2$ e si conclude.
  - Altrimenti, Eve calcola
  $$Z_2 = Z_1 - [w]G$$
  e stavolta $R_2 = [H(Z_2)]G$. Allora $H(Z_2) = k_2$ e si conclude.
- In entrambi i casi, Eve riesce a recuperare k_2, con cui calcola
$$M_2 = S_2 - [k_2]B$$

Dal cleptogramma R_2, Eve può risalire al secondo messaggio M_2 senza conoscere la chiave privata di Alice.

L'algoritmo proposto è corretto; infatti, lavorando con Z_1 e ricordando che $Y = [x]G$ si ha

$$\begin{aligned} Z_1 &= R_1 - [x]R \\ &= [k_1]G - [x]([a]R_1 + [b]G) \\ &= [k_1]G - [x][ak_1 + b]G \\ &= [k_1]G + [-ak_1 - b]Y \end{aligned} \tag{4.7}$$

Notiamo che tale valore coincide con (4.6) per $t = 0$. Quindi $Z_1 = Z$, da cui $k_2 = H(Z) = H(Z_1)$.

Se, invece, il dispositivo lavora con $t = 1$, si ha

$$\begin{aligned} Z_2 &= Z_1 - [w]G \\ &= [k_1]G + [-ak_1 - b]Y - [w]G \\ &= [k_1 - w]G + [-ak_1 - b]Y \end{aligned} \tag{4.8}$$

Anche qui ritroviamo la (4.6) per $t = 1$. Quindi $Z_2 = Z$ e $k_2 = H(Z) = H(Z_2)$.

Presentiamo un esempio concreto di come Eve possa rubare il secondo messaggio inviato da Alice.

Esempio 17.

Il primo passo è quello di concordare una curva ellittica, supponiamo $E : y^2 = x^3+2x+2$ definita sula campo $\mathbb{F}_{173}$. Notiamo che $\#E(\mathbb{F}_{173}) = 193$ il quale è un numero primo. Inoltre, come punto iniziale viene scelto $G = (2, 35)$ anch'esso di ordine $n = 193$.

Ora, Eve può generare la sua coppia di chiavi e sceglie $x = 51$ come chiave privata, da cui può calcolarsi la pubblica

$$Y = [x]G = (59, 54)$$

Alice vuole inviare due messaggi M_1 ed M_2 a Bob. Per farlo, Bob deve condividere la sua chiave pubblica

$$x_B = 8 \quad \text{e} \quad B = [x_B]G = (96, 143)$$

Quindi invia ad Alice B.

Alice vuole inviare il messaggio M_1 ="SONO". Per prima cosa codifica il testo trasformandolo in un punto di $\langle G \rangle$. Supponiamo che $M_1 = (166, 106)$. Adesso, Alice sceglie l'intero $k_1 = 123$ e calcola

$$R_1 = [k_1]G = [123]G = (41, 18)$$

e

$$S_1 = M_1 + [k_1]B = M_1 + [123]B = (54, 161)$$

Infine, Alice invia la coppia (R_1, S_1) a Bob che la utilizzerà come di consueto per recuperare il messaggio M_1.

Adesso, Alice vuole inviare il secondo messaggio M_2 ="IO" codificato con il punto $M_2 = (171, 63)$. Tuttavia, Eve si era preventivamente impossessata del dispositivo della ragazza, in cui ha inserito $a = 1$, $b = 4$ $w = 7$. Supponiamo inoltre che $t = 1$. Quando Alice genera il secondo elemento k_2, in realtà il dispositivo calcola

$$\begin{aligned} Z &= [k_1 - wt]G + [-ak_1 - b]Y \\ &= [123 - 7 \cdot 1]G + [-1 \cdot 123 - 4]Y \\ &= [116]G + [-127]Y \\ &= (96, 143) \end{aligned}$$

Di questo punto Z, viene calcolata l'impronta dell'ascissa $Z_x = 96$ tramite la funzione H. Quindi,

$$k_2 = H(Z) = H(Z_x) = H(96) = 26$$

Adesso Alice può cifrare il messaggio seguendo l'algoritmo standard. Ovvero, calcola

$$R_2 = [k_2]G = (119, 44)$$

e

$$S_2 = M_2 + [k_2]B = M_2 + [26]B = (76, 70)$$

Infine, Alice manda (R_2, S_2) a Bob.

Eve entra in scena: colleziona le coppie (R_1, S_1) ed (R_2, S_2) che sfrutterà per calcolarsi M_2. Prima di tutto calcola

$$R = [a]R_1 + [b]G = [1]R_1 + [4]G = (47, 130)$$

successivamente calcola

$$Z_1 = R_1 - [x]R = R_1 - [51]R = (11, 12)$$

Notando che $[(86, 52) = H(Z_1)G \neq R_2 = (119, 44)$, Eve conclude che deve calcolarsi Z_2. Quindi,

$$Z_2 = Z_1 - [w]G = Z_1 - [7]G = (96, 143)$$

e in effetti $Z_2 = Z$ (in realtà bastava solo che $(Z_2)_x = Z_x$).

Infine, Eve ottiene k_2 poiché

$$k_2 = H(Z) = H(Z_2) = 26$$

Conoscendo k_2 ed S_2, Eve può calcolarsi estrapolare il messaggio M_2. Infatti,

$$M_2 = S_2 - [k_2]B = (171, 63)$$

ed Eve capisce che il messaggio è "IO".

Nell'appendice B è possibile consultare i pseudocodici 6 ed 7 relativi all'implementazione del SETUP in ECEE.

Sicurezza

Un intruso, che vuole decifrare il secondo messaggio M_2 dovrebbe avere a disposizione la chiave privata di Eve, la quale non è presente nel dispositivo. Inoltre, possiamo ripercorrere per filo e per segno le dimostrazioni fatte in 4.1 per cui notiamo che il SETUP in ECEE è sicuro qualora lo sia l'ECDHP. Inoltre, l'elemento di campo nascosto Z risulta essere uniformemente distribuito in $\langle G \rangle$. Dunque, vorremmo poter concludere immediatamente che anche questo schema costituisce un SETUP forte; tuttavia il risultato non è così immediato.

A differenza del normale ECDLK, nel protocollo di cifratura di ElGamal vengono inviati due valori, R_2 ed S_2. Di per sè, il valore R_2 costituisce il cleptogramma, mentre S_2 è solo il contenitore da aprire una volta recuperata la "chiave" k_2 dal cleptogramma. Proprio questo S_2 costituisce un punto cruciale sulla sicurezza del SETUP in ECEE. Infatti, Alice, conoscendo S_2 ed altri parametri, potrebbe risalire a k_2 andando in contrasto con il punto 7 della definizione di SETUP forte 3.5. Fortunatamente per Eve, il sistema di Alice non contiene la chiave privata di Bob (la quale è semmai, nel dispositivo di quest'ultimo) e quindi risulta un problema insormontabile quello di ricavarsi k_2 da S_2. Questo ci permette di concludere che

Teorema 13. *L'attacco SETUP in ECEE è un SETUP forte iterabile ad uno schema di dispersione* $(m, m+1)$

Per verificare che tale SETUP è uno schema $(m, m+1)$ basta generare k_3 in modo tale che

$$k_3 = H(Z)$$

dove

$$Z = [k_2 - wt]G + [-ak_2 - b]Y$$

ed inviare

$$R_3 = [k_3]G \quad \text{e} \quad S_3 = M_3 + [k_3]B$$

da qui è ora possibile iterare il processo per ottenere quanto voluto.

SETUP in un crittosistema ibrido

Come già accennato nel capitolo 2, la crittografia a chiave pubblica è molto più lenta rispetto a quella simmetrica (a parità di sicurezza) per cifrare enormi messaggi. Per tale motivo, il metodo di ElGamal è spesso utilizzato in un crittosistema detto *ibrido*, dove la parte di cifratura dei messaggi è lasciata ad un protocollo simmetrico come AES o DES, mentre ElGamal si occupa solo di cifrare la chiave simmetrica (molto più esigua del messaggio in termini di bit).

Questi sistemi ibridi sono più suscettibili ad un attacco SETUP, nel senso che lo schema risultante non è più $(1, 2)$ ma addirittura $(1, 1)$ cioè, con una sola iterazione del dispositivo, Eve è in grado di recuperare il messaggio M. Vediamo velocemente come, senza divagare in dettagli.

Algoritmo

Supponiamo che Alice voglia inviare un messaggio M a Bob sfruttando il sistema ibrido ECEE-AES. Per farlo, ha bisogno di una chiave simmetrica casuale K di 128 bit che manderà a Bob sotto messaggio cifrato mediante ElGamal, il quale, a sua volta, sfrutterà un k casualmente scelto. Sia E una curva ellittica crittograficamente sicura definita sul campo $\mathbb{F}_q$. Sia G un punto iniziale di ordine un primo grande n e sia B la chiave pubblica di Bob. Eve, si era impossessata del dispositivo di Alice inserendo i soliti dati a, b, w, H, Y.

Il dispositivo di Alice compie i seguenti passi:

- Sceglie a caso un intero $k \in \{2, \ldots, n-2\}$
- Calcola l'elemento di campo nascosto
$$Z = [k - wt]G + [-ak - b]Y$$
- Infine calcola la chiave simmetrica K per AES come
$$K = H(Z_x)$$
Tale valore fungerà sia da scalare per la cifratura secondo ElGamal sia da chiave simmetrica sfruttando i 128 bit più significativi di K che denoteremo con K_{128}.
- A questo punto il dispositivo cifra la chiave simmetrica K con ECEE e il messaggio M con AES. Ovvero calcola
$$R = [K]G \quad \text{e } S = K_{128} + [K]B$$
ed invia a Bob la tripla $(AES_{K_{128}}(M), R, S)$

Eve intercetta $(AES_{K_{128}}(M), R, S)$ e compie quanto segue.

- Calcola
$$R_1 = [a]R + [b][G] \bmod p$$
- Usando la chiave privata x, calcola
$$Z_1 = R - [x]R_1$$
- A questo punto ci sono due possibilità.
  - $R = [H(Z_1)]G$, allora $H(Z_1) = K$.
  - Altrimenti, Eve calcola
$$Z_2 = Z_1 - [w]G$$
e stavolta $K = H(Z_2)$.
- In entrambi i casi Eve riesce a ricavarsi K_{128} da S con cui può recuperare il messaggio M da $AES_{K_{128}}(M)$.

Nel protocollo non contaminato, i valori k e K sono casualmente scelti al primo utilizzo del dispositivo. Implementando il SETUP, Eve va a manomettere il valore K facendolo dipendere dai parametri preinstallati installati. Il ruolo di k e K è del tutto analogo a quello di, rispettivamente, k_1 e k_2 in ECEE descritto poc'anzi, solo che quest'ultimi sono generati in non meno di due tornate del dispositivo.

4.1.3 SETUP in ECES

Sfogliando la letteratura relativa alla cleptografia su curve ellittiche non abbiamo trovato una implementazione dell'attacco SETUP nello schema di firma digitale di ElGamal in ambito ellittico. La cosa non deve sorprendere; infatti tale schema è ad oggi quasi assente nelle applicazioni, soppiantato dal più sicuro ed efficiente ECDSA a cui la somiglianza è davvero notevole. Tuttavia, per completezza, presentiamo una implementazione del SETUP anche in ECES.

Ricordiamo che gli utenti scelgono a priori la curva ellittica E ed il campo finito dove essa è definita, insieme al punto iniziale G di ordine n. Inoltre, generano anche la coppia di chiavi privata-pubblica (u, U) dove $u \in \{1, \ldots, n-1\}$ e $U = [u]G$. Notiamo che tali valori non cambieranno mai (salvo specifiche richieste da parte dell'utente o problemi tecnici). Per tale motivo, un ECDLK implementato durante la fase di generazione delle chiavi non sarebbe molto realistico, in quanto sono necessarie due chiavi per portare a termine l'attacco.

Se un utente vuole firmare un messaggio m, prima lo identifica con un intero minore di n, poi invia la tripla (m, R, s) dove $R = [k]G$ per un qualche intero k casualmente scelto, ed $s \equiv k^{-1}(m - uR_x) \bmod n$.

Algoritmo

Supponiamo che Alice voglia mandare due messaggi m_1, m_2 a Bob e li vuole firmare usando ECES. Per far ciò, acquisisce la chiave pubblica di Bob (E, q, G, B) (la funzione f non la riportiamo poiché assumiamo sia quella che restituisce l'ascissa di un punto). Sia n l'ordine di G, il quale è un primo grande.

Eve è riuscita ad impadronirsi del dispositivo di Alice e lo ha manomesso installando un meccanismo SETUP che consta dei seguenti dati: la chiave pubblica di Eve Y, tre interi a, b, w e una funzione di hash H crittograficamente sicura che genera valori minori di $n - 1$.

- La prima volta che il dispositivo viene usato compie i seguenti passi:
  - Sceglie a caso un intero $k_1 \in \{1, \ldots, n-1\}$.
  - Firma il messaggio m_1 secondo l'usuale schema ECES, quindi calcola

$$R_1 = [k_1]G \quad \text{e} \quad s_1 \equiv k_1^{-1}(m_1 - x_A f(R_1)) \bmod n$$

  - La coppia (R_1, s_1) è inviata a Bob che la usa per autenticare la provenienza del messaggio. Inoltre, k_1 è salvato in una memoria non volatile (disco rigido) in modo da essere utilizzato al secondo utilizzo dell'apparato.

- La seconda volta che il dispositivo viene usato compie i seguenti passi:
  - Sceglie uniformemente a caso un intero $t \in \{0, 1\}$.
  - Calcola

$$Z = [k_1 - w \cdot t]G + [-a \cdot k_1 - b]Y \tag{4.9}$$

– Calcola k_2 nel seguente modo

$$x_2 = H(Z)$$

(ricordiamo che per Osservazione basterebbe inviare solo l'ascissa del punto z, ovvero z_x, da cui $x_2 = H(z_x)$).

– Firma il messaggio m_2 secondo ECES, quindi calcola

$$R_2 = [k_2]G \quad \text{e} \quad s_2 \equiv k_2^{-1}(m_2 - x_A f(R_2)) \bmod n$$

Esattamente come nel caso non ellittico, Eve non deve fare altro che collezionare le coppie (R_1, s_1) e (R_2, s_2). Dopodiché, è in grado di recuperare k_2 nel seguente modo.

- Eve calcola

$$R = [a]R_1 + [b]G$$

- Usando la sua chiave privata x calcola

$$Z_1 = R_1 - [x]R$$

- A questo punto ci sono due possibilità.
  - Se $R_2 = [H(Z_1)]G$, allora $H(Z_1) = k_2$ e si conclude.
  - Altrimenti, Eve calcola
  $$Z_2 = Z_1 - [w]G$$
  e stavolta $R_2 = [H(Z_2)]G$. Allora $H(Z_2) = k_2$ e si conclude.

- In entrambi i casi, Eve riesce a recuperare k_2, con cui calcola

$$s_2 k_2 \equiv= m_2 - x_A f(R_2) \bmod n$$

Conoscendo $s_2, k_2, m_2, f(R_2)$ può tranquillamente calcolarsi la chiave privata x_A.

Dal cleptogramma R_2, Eve può risalire alla chiave x_A.

L'algoritmo proposto è corretto; infatti, lavorando con Z_1 e ricordando che $Y = [x]G$ si ha

$$\begin{aligned} Z_1 &= R_1 - [x]R \\ &= [k_1]G - [x]([a]R_1 + [b]G) \\ &= [k_1]G - [x][ak_1 + b]G \\ &= [k_1]G + [-ak_1 - b]Y \end{aligned} \tag{4.10}$$

Notiamo che tale valore coincide con (4.9) per $t = 0$. Quindi $Z_1 = Z$, da cui $k_2 = H(Z) = H(Z_1)$.

Se, invece, il dispositivo lavora con $t = 1$, si ha

$$\begin{aligned} Z_2 &= Z_1 - [w]G \\ &= [k_1]G + [-ak_1 - b]Y - [w]G \\ &= [k_1 - w]G + [-ak_1 - b]Y \end{aligned} \tag{4.11}$$

Anche qui ritroviamo la (4.9) per $t = 1$. Quindi $Z_2 = Z$ e $k_2 = H(Z) = H(Z_2)$.

Notiamo che, a differenza del caso discreto, non sono necessari aggiustamenti al protocollo poiché non sono presenti calcoli modulo un numero pari, quindi non sorgono problemi di invertibilità.

Vediamo una concreta applicazione di tale SETUP con numeri piccoli.

Esempio 18.

Il primo passo è quello di concordare una curva ellittica, supponiamo $E : y^2 = x^3+2x+2$ definita sula campo $\mathbb{F}_{173}$. Notiamo che $\#E(\mathbb{F}_{173}) = 193$ il quale è un numero primo. Inoltre, come punto iniziale viene scelto $G = (2, 35)$ anch'esso di ordine $n = 193$.

Ora, Eve può generare la sua coppia di chiavi e sceglie $x = 51$ come chiave privata, da cui può calcolarsi la pubblica

$$Y = [x]G = (59, 54)$$

Alice vuole firmare due messaggi m_1 ed m_2 ed inviarli a Bob. Per farlo, Alice deve calcolare la sua coppia di chiavi,

$$x_A = 8 \quad \text{e} \quad A = [x_A]G = (96, 143)$$

Quindi invia ad Alice B.

Alice vuole firmare il messaggio M_1 ="SONO". Per prima cosa codifica il testo trasformandolo in un intero positivo minore di n. Supponiamo che $m_1 = 166$. Adesso, Alice sceglie l'intero $k_1 = 123$ e calcola

$$R_1 = [k_1]G = [123]G = (41, 18)$$

e

$$s_1 \equiv k_1^{-1}(m_1 - x_A \cdot (R_1)_x) \equiv 102 \cdot (166 - 8 \cdot 41) \equiv 74 \bmod 193$$

Infine, Alice invia il messaggio m_1 insieme alla coppia (R_1, s_1) a Bob che la utilizzerà come di consueto per verificare l'autenticità della firma.

Adesso, Alice vuole firmare il secondo messaggio m_2 ="IO" codificato con l'intero $m_2 = 171$. Tuttavia, Eve si era preventivamente impossessata del dispositivo della ragazza, in cui ha inserito $a = 1$, $b = 4$ $w = 7$. Supponiamo inoltre che $t = 1$. Quando Alice genera il secondo elemento k_2, in realtà il dispositivo calcola

$$\begin{aligned} Z &= [k_1 - wt]G + [-ak_1 - b]Y \\ &= [123 - 7 \cdot 1]G + [-1 \cdot 123 - 4]Y \\ &= [116]G + [-127]Y \\ &= (96, 143) \end{aligned}$$

Di questo punto Z, viene calcolata l'impronta dell'ascissa $Z_x = 96$ tramite la funzione H. Quindi,

$$k_2 = H(Z) = H(Z_x) = H(96) = 26$$

Adesso Alice può cifrare il messaggio seguendo l'algoritmo standard. Ovvero, calcola

$$R_2 = [k_2]G = (119, 44)$$

e

$$s_2 \equiv k_2^{-1}(m_2 - x_A(R_2)_x) \equiv 52 \cdot (171 - 8 \cdot 119) \equiv 111 \bmod 193$$

Infine, Alice manda (R_2, s_2) ed il messaggio m_2 a Bob.

Eve entra in scena: colleziona i messaggi e le relative coppie (R_1, s_1) ed (R_2, s_2), che sfrutterà per calcolarsi la chiave privata di Alice x_A. Prima di tutto calcola

$$R = [a]R_1 + [b]G = [1]R_1 + [4]G = (47, 130)$$

successivamente calcola

$$Z_1 = R_1 - [x]R = R_1 - [51]R = (11, 12)$$

Notando che $[(86, 52) = H(Z_1)G \neq R_2 = (119, 44)$, Eve conclude che deve calcolarsi Z_2. Quindi,

$$Z_2 = Z_1 - [w]G = Z_1 - [7]G = (96, 143)$$

e in effetti $Z_2 = Z$ (in realtà bastava solo che $(Z_2)_x = Z_x$).

Infine, Eve ottiene k_2 poiché

$$k_2 = H(Z) = H(Z_2) = 26$$

Conoscendo k_2 ed s_2, Eve può estrapolare x_A. Infatti,

$$x_A \equiv (R_2)_x^{-1}(m_2 - s_2 k_2) \equiv 133 \cdot (171 - 111 * 26) \equiv 8 \bmod 193$$

Nell'appendice B è possibile consultare i pseudocodici 9 ed 10 relativi all'implementazione del SETUP in ECES.

Sicurezza

Un intruso, per risalire alla chiave privata di Alice x_A dovrebbe avere a disposizione la chiave x di Eve, la quale non è sita nel dispositivo. Inoltre, possiamo ripercorrere per filo e per segno le dimostrazioni fatte in 4.1 per cui notiamo che il SETUP in ECES è sicuro qualora lo sia l'ECDHP. In più, l'elemento di campo nascosto Z risulta essere uniformemente distribuito in $\langle G \rangle$. Dunque, vorremmo poter concludere immediatamente che anche questo schema costituisce un SETUP forte; tuttavia non è così.

Teorema 14. *L'attacco SETUP implementato nello schema di firma di ElGamal su curve ellittiche costituisce un SETUP regolare ed è uno schema di dispersione* $(1, m)$ *con* $m \geq 2$.

Dimostrazione. I punti 1-6 della definizione 3.2 restano ancora validi purché i valori a, b e il seme di H restino segreti, tuttavia viene meno il settimo punto della definizione 3.5. Infatti, ricordiamo che Alice conosce l'effettiva implementazione del suo dispositivo e può analizzarlo prima e dopo ogni utilizzo. Quindi, la ragazza può conoscere la sua chiave privata x_A, con cui può risalire a k_1 e k_2 semplicemente calcolando

$$k_i \equiv s_i^{-1}(m_i - a(R_i)_x) \bmod n$$

Con questo metodo è in grado di recuperare tutti gli esponenti passati. Se, inoltre, conosce a, b e il seme di H, allora Alice può definitivamente rilevare la presenza di un meccanismo SETUP nel suo dispositivo. Infatti, sotto tali condizioni, ella è in grado di calcolarsi H^{k_2} e Z e di confrontarli.

Lo schema è $(1, 2)$ poiché con due iterazioni del dispositivo Eve può risalire alla chiave privata x_A. Tuttavia, qualora Alice volesse firmare un terzo messaggio m_3, il dispositivo contaminato calcolerebbe

$$k_3 = H(Z)$$

dove

$$Z = [k_2 - wt]G + [-ak_2 - b]Y$$

ed invierebbe a Bob il messaggio m_3 più la firma (R_3, s_3) dove

$$R_3 = [k_3]G \quad \text{e} \quad s_3 \equiv k_3^{-1}(m_3 - x_A(R_2)_x) \bmod n$$

Da qui, Eve può ricavarsi k_3 che sfrutta per trovarsi ancora una volta x_A.

Utilizzando m volte il dispositivo, Eve risale sempre e comunque alla sola chiave x_A. Ovvero il SETUP è uno schema $(1, m)$. □

4.1.4 SETUP in ECDSA

Riuscire ad installare una backdoor asimmetrica in ECDSA non è un risultato di poco conto. Infatti, come già detto esso è il principale schema di firma digitale attualmente in circolazione; utilizzato da Bitcoin e da Sony e da tante altre aziende. Quindi, vediamo come si implementa il SETUP in ECDSA.

Ricordiamo che gli utenti scelgono a priori una curva ellittica E definita sul campo $\mathbb{F}_q$ ed una funzione di hash h. Sia G il punto iniziale di ordine un primo n. Gli utenti generano anche la coppia di chiavi privata-pubblica (u, U) dove $u \in \{1, \ldots, n-1\}$ e $U = [u]G$. Notiamo che tali valori non cambieranno mai (salvo specifiche richieste da parte dell'utente o problemi tecnici). Per tale motivo, un ECDLK implementato durante la fase di generazione delle chiavi non sarebbe molto realistico, in quanto sono necessarie due chiavi per portare a termine l'attacco.

Se un utente vuole firmare un messaggio m, prima lo identifica con un intero minore di n, poi invia la tripla (m, R, s) dove $R = [k]G$ per un qualche intero k casualmente scelto, ed $s \equiv k^{-1}(h(m) + uR_x) \bmod n$.

Algoritmo

Supponiamo che Alice voglia mandare due messaggi m_1, m_2 a Bob e il vuole firmare usando ECDSA. Per far ciò, acquisisce la chiave pubblica di Bob (E, q, G, B) (la funzione f non la riportiamo poiché è quella che restituisce l'ascissa di un punto). Sia n l'ordine di G, il quale è un primo grande. Infine, sia x_A la chiave privata di Alice.

Eve è riuscita ad impadronirsi del dispositivo di Alice e lo ha manomesso installando un meccanismo SETUP che consta dei seguenti dati: la chiave pubblica di Eve Y, tre interi a, b, w e una funzione di hash H crittograficamente sicura che genera valori minori di $n - 1$.

- La prima volta che il dispositivo viene usato compie i seguenti passi:
  - Sceglie a caso un intero $k_1 \in \{1, \ldots, n-1\}$.
  - Firma il messaggio m_1 secondo l'usuale schema ECDSA, quindi calcola
  $$R_1 = [k_1]G \quad \text{e} \quad s_1 \equiv k_1^{-1}(h(m_1) + x_A f(R_1)) \bmod n$$
  - La coppia (R_1, s_1) è inviata a Bob che la usa per autenticare la provenienza del messaggio. Inoltre, k_1 è salvato in una memoria non volatile (disco rigido) in modo da essere utilizzato al secondo utilizzo dell'apparato.
- La seconda volta che il dispositivo viene usato compie i seguenti passi:
  - Sceglie uniformemente e a caso un intero $t \in \{0, 1\}$.
  - Calcola
  $$Z = [k_1 - w \cdot t]G + [-a \cdot k_1 - b]Y \tag{4.12}$$
  - Calcola k_2 nel seguente modo
  $$x_2 = H(Z)$$
  - Firma il messaggio m_2 secondo ECDSA, ovvero calcola
  $$R_2 = [k_2]G \quad \text{e} \quad s_2 \equiv k_2^{-1}(h(m_2) + x_A f(R_2)) \bmod n$$

Esattamente come nel caso non ellittico, Eve non deve fare altro che collezionare le coppie (R_1, s_1) e (R_2, s_2). Dopodiché, è in grado di recuperare k_2 nel seguente modo.

- Eve calcola
$$R = [a]R_1 + [b]G$$
- Usando la sua chiave privata x calcola
$$Z_1 = R_1 - [x]R$$
- A questo punto ci sono due possibilità.

– Se $R_2 = [H(Z_1)]G$, allora $H(Z_1) = k_2$ e si conclude.
– Altrimenti, Eve calcola
$$Z_2 = Z_1 - [w]G$$
e stavolta $R_2 = [H(Z_2)]G$. Allora $H(Z_2) = k_2$ e si conclude.

- In entrambi i casi, Eve riesce a recuperare k_2, con cui calcola
$$s_2 k_2 \equiv= h(m_2) + x_A f(R_2) \bmod n$$
Conoscendo $s_2, k_2, h(m_2), f(R_2)$ può tranquillamente calcolarsi la chiave privata x_A.

Dal cleptogramma r_2, Eve può risalire alla chiave x_A.

L'algoritmo proposto è corretto; infatti, lavorando con Z_1 e ricordando che $Y = [x]G$ si ha

$$\begin{aligned} Z_1 &= R_1 - [x]R \\ &= [k_1]G - [x]([a]R_1 + [b]G) \\ &= [k_1]G - [x][ak_1 + b]G \\ &= [k_1]G + [-ak_1 - b]Y \end{aligned} \tag{4.13}$$

Notiamo che tale valore coincide con (4.6) per $t = 0$. Quindi $Z_1 = Z$, da cui $k_2 = H(Z) = H(Z_1)$.

Se, invece, il dispositivo lavora con $t = 1$, si ha

$$\begin{aligned} Z_2 &= Z_1 - [w]G \\ &= [k_1]G + [-ak_1 - b]Y - [w]G \\ &= [k_1 - w]G + [-ak_1 - b]Y \end{aligned} \tag{4.14}$$

Anche qui ritroviamo la (4.5) per $t = 1$. Quindi $Z_2 = Z$ e $k_2 = H(Z) = H(Z_2)$.

Vediamo con un esempio, come Eve riesce a rubare la chiave privata di Alice x_A.

Esempio 19.

- Il primo passo è quello di concordare una curva ellittica, supponiamo $E : y^2 = x^3 + 2x + 2$ definita sula campo $\mathbb{F}_{173}$. Notiamo che $\#E(\mathbb{F}_{173}) = 193$ il quale è un numero primo. Inoltre, come punto iniziale viene scelto $G = (2, 35)$ anch'esso di ordine $n = 193$.

- Ora, Eve può generare la sua coppia di chiavi e sceglie $x = 51$ come chiave privata, da cui può calcolarsi la pubblica
$$Y = [x]G = (59, 54)$$

- Alice vuole firmare due messaggi m_1 ed m_2 ed inviarli a Bob. Per farlo, Alice deve calcolare la sua coppia di chiavi,
$$x_A = 8 \quad \text{e} \quad A = [x_A]G = (96, 143)$$
Quindi invia ad Alice B.

- Alice vuole firmare il messaggio M_1 ="SONO". Per prima cosa codifica il testo mediante una funzione di hash h in un intero positivo minore di n. Supponiamo che $h(m_1) = 166$. Adesso, Alice sceglie l'intero $k_1 = 123$ e calcola

$$R_1 = [k_1]G = [123]G = (41, 18)$$

e

$$s_1 \equiv k_1^{-1}(h(m_1) + x_A \cdot (R_1)_x) \equiv 102 \cdot (166 - 8 \cdot 41) \equiv 15 \bmod 193$$

Infine, Alice invia il messaggio $h(m_1)$ insieme alla coppia (R_1, s_1) a Bob che la utilizzerà come di consueto per verificare l'autenticità della firma.

- Adesso, Alice vuole firmare il secondo messaggio m_2 ="IO" codificato con l'intero $h(m_2) = 171$. Tuttavia, Eve si era preventivamente impossessata del dispositivo della ragazza, in cui ha inserito $a = 1$, $b = 4$ $w = 7$. Supponiamo inoltre che $t = 1$. Quando Alice genera il secondo elemento k_2, in realtà il dispositivo calcola

$$\begin{aligned} Z &= [k_1 - wt]G + [-ak_1 - b]Y \\ &= [123 - 7 \cdot 1]G + [-1 \cdot 123 - 4]Y \\ &= [116]G + [-127]Y \\ &= (96, 143) \end{aligned}$$

Di questo punto Z, viene calcolata l'impronta dell'ascissa $Z_x = 96$ tramite la funzione H. Quindi,

$$k_2 = H(Z) = H(Z_x) = H(96) = 26$$

Adesso Alice può cifrare il messaggio seguendo l'algoritmo standard. Ovvero, calcola

$$R_2 = [k_2]G = (119, 44)$$

e

$$s_2 \equiv k_2^{-1}(h(m_2) + x_A(R_2)_x) \equiv 52 \cdot (171 - 8 \cdot 119) \equiv 110 \bmod 193$$

Infine, Alice manda (R_2, s_2) ed il messaggio $h(m_2)$ a Bob.

- Eve entra in scena: colleziona i messaggi e le relative coppie (R_1, s_1) ed (R_2, s_2), che sfrutterà per calcolarsi la chiave privata di Alice x_A. Prima di tutto calcola

$$R = [a]R_1 + [b]G = [1]R_1 + [4]G = (47, 130)$$

successivamente calcola

$$Z_1 = R_1 - [x]R = R_1 - [51]R = (11, 12)$$

Notando che $[(86, 52) = H(Z_1)G \neq R_2 = (119, 44)$, Eve conclude che deve calcolarsi Z_2. Quindi,

$$Z_2 = Z_1 - [w]G = Z_1 - [7]G = (96, 143)$$

e in effetti $Z_2 = Z$ (in realtà bastava solo che $(Z_2)_x = Z_x$).

Infine, Eve ottiene k_2 poiché

$$k_2 = H(Z) = H(Z_2) = 26$$

- Conoscendo k_2 ed s_2, Eve può estrapolare x_A. Infatti,

$$x_A \equiv (R_2)_x^{-1}(s_2k_2 - h(m_2)) \equiv 133 \cdot (111 * 26 - 171) \equiv 8 \bmod 193$$

Nell'appendice B è possibile consultare i pseudocodici 12 ed 13 relativi all'implementazione del SETUP in ECDSA.

Sicurezza

I protocolli ECDSA ed ECES si somigliano significativamente. Non deve sorprendere quindi, che anche le trattazioni della sicurezza dei relativi SETUP lo siano. Per completezza, riportiamo quanto già detto in 4.1.3.

Un intruso, per risalire alla chiave privata di Alice x_A dovrebbe avere a disposizione la chiave x di Eve, la quale non è sita nel dispositivo. Inoltre, possiamo ripercorrere per filo e per segno le dimostrazioni fatte in 4.1 per cui notiamo che il SETUP in ECDSA è sicuro qualora lo sia l'ECDHP. In più, l'elemento di campo nascosto Z risulta essere uniformemente distribuito in $\langle G \rangle$. Dunque, vorremmo poter concludere immediatamente che anche questo schema costituisce un SETUP forte; tuttavia non è così.

Teorema 15. *L'attacco SETUP implementato nello schema ECDS costituisce un SETUP regolare ed è uno schema* $(1, m)$ *con* $m \geq 2$.

Dimostrazione. I punti 1-6 della definizione 3.2 restano ancora validi purché i valori a, b e il seme di H restino segreti, tuttavia viene meno il settimo punto della definizione 3.5. Infatti, ricordiamo che Alice conosce l'effettiva implementazione del suo dispositivo e può analizzarlo prima e dopo ogni utilizzo. Quindi, la ragazza può conoscere la sua chiave privata x_A, con cui può risalire a k_1 e k_2 semplicemente calcolando

$$k_i \equiv s_i^{-1}(h(m_i) + a(R_i)_x) \bmod n$$

Con questo metodo è in grado di recuperare tutti gli esponenti passati. Se, inoltre, conosce a, b e il seme di H, allora Alice può definitivamente rilevare la presenza di un meccanismo SETUP nel suo dispositivo. Infatti, sotto tali condizioni, ella è in grado di calcolarsi H^{k_2} e Z e di confrontarli.

Lo schema è $(1, 2)$ poiché con due iterazioni del dispositivo Eve può risalire alla chiave privata x_A. Tuttavia, qualora Alice volesse firmare un terzo messaggio m_3, il dispositivo contaminato calcolerebbe

$$k_3 = H(Z)$$

dove

$$Z = [k_2 - wt]G + [-ak_2 - b]Y$$

ed invierebbe a Bob il messaggio m_3 più la firma (R_3, s_3) dove

$$R_3 = [k_3]G \quad \text{e} \quad s_3 \equiv k_3^{-1}(h(m_3) + x_A(R_2)_x) \bmod n$$

Da qui, Eve può ricavarsi k_3 che sfrutta per trovarsi ancora una volta x_A.

Utilizzando m volte il dispositivo, Eve risale sempre e comunque alla sola chiave x_A. Ovvero il SETUP è uno schema $(1, m)$. □

4.2 Considerazioni sulla complessità degli algoritmi

Nelle sezioni precedenti abbiamo visto come è possibile implementare un attacco SETUP in sistemi basati sul ECDLP. Tuttavia, nella vita reale, sorgono problemi relativi alla complessità degli algoritmi contaminati. Infatti, i SETUP altro non sono che delle modifiche apportate agli algoritmi sottostanti aggiungendo ulteriori parametri e operazioni. Proprio queste aggiunte vanno poste sotto la lente di ingrandimento perché potrebbero variare notevolmente la complessità computazionale, rendendo inutile un qualsiasi utilizzo di un attacco SETUP. Fortunatamente, a livello asintotico ciò non avviene.

Consideriamo il SETUP in ECDH 4.1.1 e confrontiamolo con l'algoritmo sano. Per farlo faremo uso dei pseudocidici rispettivamente 3 e 2. Tralasciando varie assegnazioni e confronti che possiamo considerare come $O(1)$, l'operazione più pesante è la moltiplicazione scalare. Nel sistema contaminato abbiamo tre operazioni scalari (riga 7 e 9) contro l'unica eseguita da quello sano (riga 2). Assumendo che gli scalari siano, in termini di bit, minori o uguali a q, allora l'algoritmo sano ha una complessità computazionale pari a $O(\log^3 q)$, mentre quello contaminato $(3 \cdot \log^3 q) = O(\log^3 q)$. Dunque, i due sistemi hanno la stessa complessità computazionale. Anche per gli altri protocolli è possibile osservare che il rapporto tra le moltiplicazioni scalari è sempre 3 a 1 indipendentemente dai bit in ingresso, tranne che per ECEE dove il rapporto è 4 a 2. Quindi, per tutti i sistemi analizzati, la complessità computazionale degli algoritmi contaminati coincide cone quella degli algoritmi sani. In effetti, questa analisi ricalca il secondo punto della definizione di SETUP 3.2 in cui viene chiesto che il sistema contaminato C' debba lavorare efficacemente tanto quanto il sistema sano C.

Vale la pena notare che anche il recupero delle informazioni private da parte de Eve avviene con complessità polinomiale in q. In effetti, sono sempre presenti (al più) 5 prodotto scalari e queste costituiscono la parte più pesante dell'algoritmo, In totale si hanno $O(5 \cdot \log^3 q) = O(\log^3 q)$ operazioni elementari.

Attacco a cronometro

Ricordiamo che la notazione O-grande fornisce solo una stima asintotica del reale tempo di esecuzione di un algoritmo in termini dei valori di ingresso. In concreto, quelle due moltiplicazioni scalari in più nei sistemi contaminati dal SETUP hanno un peso specifico non trascurabile nell'analisi di tale tempo di esecuzione. In effetti, in computer comuni da circa 1.80 GHz per core, la moltiplicazione scalare richiede circa 10 ms, se ne vengono eseguite altre due, il tempo si dilata a circa 30 ms. Questi sono valori facilmente rivelabili e confrontabili e possono costituire un difficile ostacolo da superare nella concreta implementazione di un SETUP. Ad, esempio, Alice potrebbe generare una moltitudine

di chiavi, misurare il tempo necessario per compiere tali operazioni, e confrontare i dati ottenuti con quelli di un dispositivo analogo certificato come non manomesso. In caso di differenze sostanziali, Alice potrebbe insospettirsi e decidere di non utilizzare più il suo device. Tale metodo di analisi prende il nome di *attacco a cronometro* (o *timing attack*) il quale rientra nella più ampia categoria dei *canali laterali*, ovvero attacchi basati sull'implementazione dei crittositemi piuttosto che sulle debolezze degli algoritmi dell'implementazione stessa. Dunque, essi analizzano fattori "esterni" quali il tempo impiegato appunto, ma anche il consumo energetico, i suoni prodotti, le radiazioni elettromagnetiche emesse ad altri ancora.

In figura 4.1 sono riportati quattro grafici utili ad analizzare i tempi di esecuzioni degli algoritmi studiati nella sezione 2.5 e compararli con gli analoghi contaminati. Le figure sono state generate usando Python 3.8 su piattaforma Windows da 64 bit dotato di quattro core da 1.80 GHz ciascuno e una RAM da 6 Gb. La curva ellittica sottostante è la *secp256k1* i cui dettagli sono disponibili in [36].

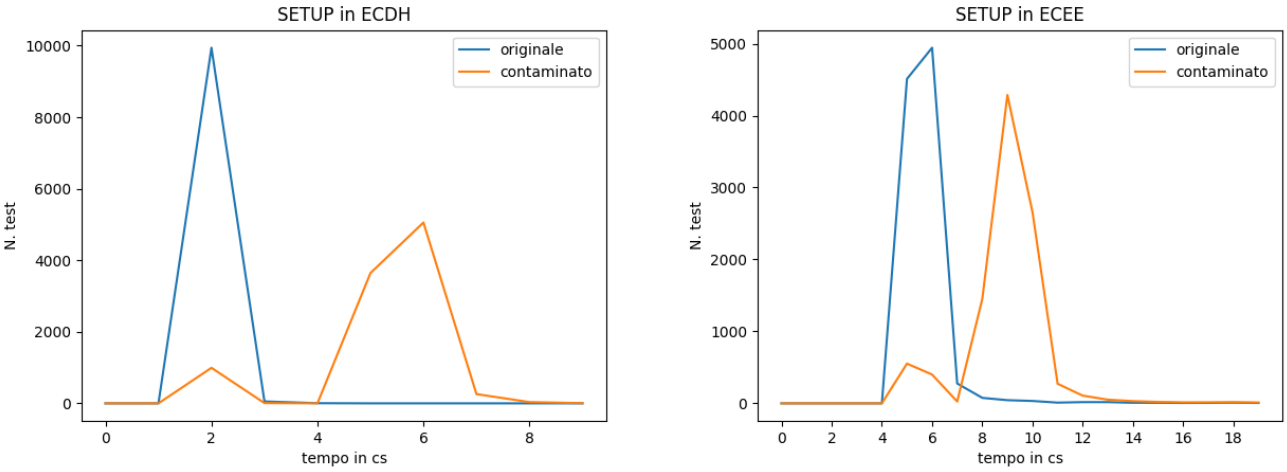
In tutte e quattro le figure compaiono dei piccoli picchi nello stesso valore temporale in cui è presente il picco principale dell'algoritmo non contaminato. Questo è dovuto al fatto che, per ogni test, vengono eseguite 10 iterazioni del dispositivo. La prima iterazione si comporta esattamente come l'algoritmo sano, le altre 9 fanno riferimento alla parte aggiuntiva relativa all'elemento di campo nascosto, quella descritta dopo la frase "La seconda volta che il dispositivo viene usato".

I grafici 4.1a, 4.1c 4.1d sono molto simili tra loro, questo perché tutti e tre gli algoritmi sani (ECDH, ECES, ECDSA) performano una sola moltiplicazione scalare, mentre quelli contaminati ne compiono tre. Tuttavia, per ECEE (grafico 4.1b) il rapporto tra moltiplicazioni scalari "contaminate" e "sane" è 4 a 2. Ciò rende il SETUP in ECEE quello più plausibile da implementare in applicazioni concrete, ma ancora semplice da individuare da parte di un utente che decide di verificare la verginità di un dispositivo mediante un attacco a cronometro.

In realtà, i grafici riportati dipendono fortemente dalla capacità del processore usato; CPU più potenti potrebbero rappresentare gli stessi picchi ma molto più vicini tra loro in termini di centisecondi, rendendo, all'apparenza, i SETUP molto più efficaci di quel che non appaiono in questa tesi. Per tale motivo è opportuno considerare il *coefficiente di variazione* CoV, il quale misura la dispersione dei dati attorno al valore medio (quello dei picchi sostanzialmente) indipendentemente dalla capacità del processore. Quindi, il CoV di un test eseguito su un computer da 1.80 GHz è lo stesso di quello fornito da un processore da 4 GHz per esempio. Formalmente parlando, dato un campione di dati, il coefficiente di varianza è il rapporto tra la deviazione standard σ ed il valor medio E di questi dati, ovvero

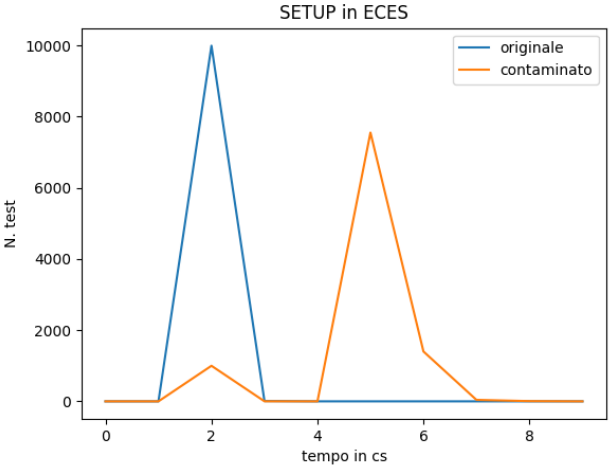
$$CoV = \frac{\sigma}{E}$$

Nel nostro contesto, il calcolo di CoV è basato sulla differenza del numero di moltiplicazioni scalari tra un algoritmo sano e quello contaminato. Per esempio, supponiamo che in un algoritmo sano il costo della moltiplicazione scalare ha varianza σ^2 e valor medio E, allora $CoV_1 = \sigma/E$ (ricordiamo che la deviazione standard è la radice quadrata della varianza). Se l'algoritmo contaminato consta di tre moltiplicazioni scalari, allora il costo totale ha varianza $3\sigma^2$ e tempo medio $3E$. In questo caso $CoV_2 = \sqrt{3}\sigma/3E = \sigma/\sqrt{3}E$.

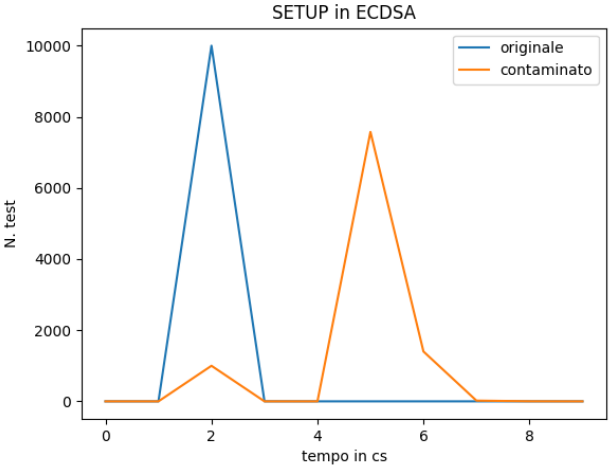


(a)

(b)



(c)



(d)

Figura 4.1: Tempi di esecuzione degli algoritmi sani e contaminati.

Per conoscere la vulnerabilità di un meccanismo SETUP contro un attacco a cronometro, basta calcolare il rapporto R tra il CoV dell'algoritmo sano e di quello contaminato. Tanto più questo rapporto si avvicina ad uno, tanto è più difficile rilevare la presenza del SETUP. Continuando l'esempio sopra, avremo che $R = CoV_1/CoV_2 = \sqrt{3} \approx 1.73$.

Riportiamo nella tabella 4.1 i valori di questi rapporti per i quattro algoritmi proposti.

	ECDH	ECEE	ECES	ECDSA
R	1.73	1.41	1.73	1.73

Tabella 4.1: Rapporti dei coefficienti di variazione.

Quindi, anche a livello teorico, è più difficile rilevare la presenza di un SETUP in ECEE piuttosto che in ECDH, ECES, ECDSA.

Attualmente, non si conoscono implementazioni di attacchi SETUP relativi agli algoritmi descritti in questa tesi che siano irrintracciabili mediante analisi del tempo di esecuzione degli algoritmi. Tuttavia, Young e Yung risolsero questo problema per il SETUP in RSA, riuscendo a progettare una backdoor addirittura più veloce della normale generazione delle chiavi in RSA [52].

Infine, teniamo a precisare che tale metodo è volto a misurare, in un certo senso, l'efficacia in una applicazione reale di tali attacchi. Tuttavia, è possibile seguire ulteriori vie, fornendo, per l'appunto, una vera e propria metrica. In questo senso, citiamo il recentissimo lavoro di Kovalenko e Kudin riguardo il *potenziale cleptografico* [21].

Capitolo 5

Conclusione

In questo capitolo finale riassumeremo brevemente i risultati salienti presentati in questa tesi. Quindi, descriveremo il DLK e costruiremo una tabella in cui riportiamo a quale tipologia di SETUP appartengono gli algoritmi analizzati (debole, regolare o forte).

5.1 Riepilogo attacchi SETUP

Abbiamo visto come è possibile implementare un attacco SETUP nei protoclli di Diffie-Hellman, di ElGamal e nel DSA e nelle rispettive varianti ellittiche. Tutti questi attacchi hanno una radice comune che è il Discrete Log Kleptogram. In sintesi, tale schema utilizza due esecuzioni consecutive del dispositivo contaminato che al suo interno incorpora la chiave privata dell'attaccante ed altri parametri sviatori. Il malintenzionato ricerca nell'algoritmo sottostante un elevamento alla potenza $g^k \bmod p$ che sia ricavabile dagli output del dispositivo. Tale valore fungerà da cleptogramma poiché, al secondo utilizzo del device, il nuovo esponente k' dipenderà dai parametri privati dell'assalitore precedentemente menzionati. Conoscendo la struttura di k' e collezionando gli output, l'attaccante può ricavarsi le informazioni private della vittima.

Analogamente funziona il ECDLK solo che l'elevamento alla potenza è sostituito con una moltiplicazione scalare $[k]G$.

Riassumiamo nella tabella 5.1 la natura dei SETUP presentati durante il corso della tesi, precisando anche la leakage bandwidth e il cleptogramma sfruttato.

Per ciò che concerne gli attacchi SETUP implementati nelle versioni ellittiche, abbiamo presentato anche una possibile tecnica per rilevare la presenza di un tale meccanismo all'interno del dispositivo. L'idea è quella di misurare il tempo di esecuzione del dispositivo (contaminato) e di confrontarlo con uno la cui integrità è rassicurata da un

SETUP in	Tipo	Leakage bandwidth	cleptogramma
DH / ECDH	forte	$(m, m+1)$	$y_2 \equiv g^{x_2} \bmod p$ / $A_2 = [x_2]G$
Cifratura di ElGamal / ECEE	forte	$(m, m+1)$	$r_2 \equiv g^{k_2} \bmod p$ / $R_2 = [k_2]G$
Firma di ElGamal / ECES	regolare	$(1, m)$	$r_2 \equiv g^{k_2} \bmod p$ / $R_2 = [k_2]G$
DSA / ECDSA	regolare	$(1, m)$	$g^{k_2} \bmod p$ / $R_2 = [k_2]G$

Tabella 5.1: Tabella riassuntiva dei vari SETUP.

ente certificatore. Qualora riscontriamo anomalie nel conteggio temporale, possiamo concludere che il dispositivo è stato precedentemente manomesso. Analizzando il divario temporale tra un dispositivo sano ed uno contaminato per mezzo del coefficiente di variazione, abbiamo mostrato come, tra i quattro algoritmi proposti, il SETUP in ECEE è probabilmente quello più difficile da rilevare mediante un attacco a cronometro. Ciò rende questo attacco il più efficacie da implementare per applicazioni concrete. Nonostante ciò, riteniamo che siano possibili migliorie agli algoritmi in modo da renderli ancora più identificabili mediante un check di integrità. Ecco alcuni suggerimenti.

- Grazie all'osservazione 7 è possibile semplificare gli algoritmi togliendo almeno una moltiplicazione scalare, ovvero, sarebbe possibile costruire l'elemento di campo nascosto Z così:

  $$Z = [k]Y$$

  Ovviamente sorgerebbero piccoli problemi relativi alla effettiva sicurezza dei vari SETUP, ma ricordiamo che tali problematiche sono strettamente legate alla possibilità di invertire la funzione di hash H. Se tale funzione è ben scelta, la sua invertibilità è un problema di complessità paragonabile a quella di un DLP, e quindi di difficile risoluzione.

- Una ulteriore idea è quella di inserire un "interruttore" casuale che accende e spegne il SETUP all'interno del dispositivo. Con tale meccanismo, la probabilità che un dispositivo contaminato passi un check di integrità è molto più elevata. Tuttavia, va considerato che il malintenzionato che ha manomesso il device dovrebbe essere in grado di capire quando un attacco SETUP è in esecuzione in modo da poter collezionare gli opportuni output.

- Come già detto, le moltiplicazioni scalari e le esponeziazioni sono le operazioni dal costo più elevato all'interno degli algoritmi proposti. Una idea, quindi, sarebbe quella di implementare un attacco SETUP che non fa uso di tali operazioni (o almeno in numero strettamente limitato). Questa ricerca supponiamo non sia affatto semplice, in primo luogo perché dei nuovi SETUP dovrebbero sostituire i già collaudati DLK ed ECDLK, in seconda analisi perché si vorrebbe mantenere un grado di sicurezza pari se non superiore a quello fornito da DLK ed ECDLK. Tuttavia, un primo passo verso questa direzione è fornita dagli stessi Young e Yung nel loro libro "*Malicious Cryptography: Exposing Cryptovirology*" [54].

5.2 Ultime considerazioni

Il lavoro iniziato da Young e Yung nel 1996 è ancora molto giovane ma eccellenti risultati si sono raggiunti specialmente per quanto riguarda l'implementazione di un SETUP in RSA. Per tale motivo abbiamo preferito dare voce ai meno chiacchierati DH, ElGamal ed DSA. In particolare, ci siamo soffermati sulle varianti ellittiche di questi schemi, poiché esse, stanno trovando maggior diffusione nelle applicazioni reali.

Sottolineiamo che tale tesi è volta a presentare in tutta generalità il meccanismo SETUP senza soffermarsi su come difendersi da questo attacco. Per chi volesse intraprendere questa strada consigliamo la lettura di un articolo pubblicato da Kovalenko e Kudin [22]

i quali proposero più modi per ampliare la sicurezza di RSA e DH in modo da renderli invulnerabili (o quasi) a questo attacco cleptografico.

Infine, vogliamo concludere con un pensiero riguardante la massiccia digitalizzazione del mondo moderno. Solitamente siamo utilizzatori inerti degli strumenti con cui interagiamo tutti i giorni (smartphone, WhatsApp, Google ecc.) il che li rende dei dispositivi a scatola nera per i nostri occhi inesperti. Gli articoli di Young e Yung hanno evidenziato i rischi di questa nostra passività, ma il primo vero ed importante passo per superarla deve venire da noi utenti comuni, in quanto utilizzatori finali di tali dispositivi. Dobbiamo prendere consapevolezza di ciò che stiamo usando e porci delle domande quali "E' sicuro?", "Mi posso fidare di chi me lo ha dato?", "Riesco a capire almeno un po' come funziona?". Appena qualcosa va storto è facile incolpare gli altri della nostra ignoranza, ma, citando il poeta Kahlil Gibran:

> Se riveli al vento i tuoi segreti, non devi poi rimproverare al vento di rivelarli agli alberi.

Appendice A

Ulteriori elementi di crittografia

In questa appendice presentiamo ulteriori nozioni quali funzioni di hash, funzioni direzionali, canali subliminali, utili al fine di una più chiara comprensione del testo.

A.1 Canali Subliminali

Definizione A.1. Un *canale subliminale* è un metodo di comunicazione nascosto all'interno di un protocollo crittografico. Tale metodo consiste nell'inviare messaggi aggiuntivi latenti nel messaggio principale, tali che non possano essere letti da altri utenti al di fuori del destinatario.

Il pioniere dei canali subliminali fu Gustavus Simmons che per primo li introdusse nel 1984 come soluzione al *Problema dei Prigionieri* [45]. Riportiamo il testo originale per invitare il lettore a cimentarsi in questo indovinello.

> Two accomplices in a crime have been arrested and are about to be locked in widely separated cells. Their only means of communication after they are locked up will he by way of messages conveyed for them by trustees – who are known to be agents of the warden. The warden is willing to allow the prisoners to exchange messages in the hope that he can deceive at least one of them into accepting as a genuine communication from the other either a fraudulent message created by the warden himself or else a modification by him of a genuine message. However, since he has every reason to suspect that the prisoners want to coordinate an escape plan, the warden will only permit the exchanges to occur if the information contained in the messages is completely open to him – and presumably innocuous. The prisoners, on the other hand, are willing to accept these conditions, i.e., to accept some risk of deception in order to be able to communicate at all, since they need to coordinate their plans. To do this they will have to deceive the warden by finding a way of communicating secretly in the exchanges, i.e., of establishing a "subliminal channel" between them in full view of the warden, even though the messages themselves contain no secret (to the warden) information. Since they anticipate that the warden will try to deceive them by introducing fraudulent messages, they will only exchange messages if they are permitted to authenticate them.

Un facile esempio di canale subliminale è basato sul numero di parole presenti in un messaggio. Un numero pari di parole significa 0; un numero dispari significa 1.

Un altro esempio è quello di celare l'informazione aggiuntiva sfruttando le lettere iniziali di ciascuna parola. In questo modo, Alice e Bob possono scambiarsi ulteriori messaggi purché qualcuno non scopra il metodo utilizzato.

In realtà, è auspicabile che la malintenzionata Eve sappia dell'esistenza di un canale di comunicazione secondario tra Alice e Bob. Per questo i due ragazzi devono sviluppare un metodo che non permetta ad Eve di leggere i messaggi. A tal proposito, ritorniamo al problema dei prigionieri e presentiamo ora una soluzione descritta da Young e Yung [54] (ovviamente Alice e Bob sono i prigionieri, mentre Eve è la guardia).

Supponiamo che Alice voglia mandare un messaggio a Bob e assumiamo che i due si siano messi preventivamente d'accordo su un chiave di firma segreta x, prima di essere arrestati da Eve.

- Alice genera un messaggio innocuo e palese.
- Alice firma il messaggio usando la chiave x in modo da nascondere il contenuto subliminale nella firma.
- Alice manda il messaggio ad Eve, affinché lo controlli.
- Eve verifica la firma di Alice; non trovando nulla di compromettente, invia il messaggio a Bob.
- Bob ignora il messaggio e controlla la firma di Alice, se è valida passa al punto successivo, altrimenti sa che il messaggio è compromesso.
- Bob usa la chiave segreta x per recuperare il messaggio nascosto.

La guardia Eve non ha la possibilità di accorgersi con criterio logico dell'aggiramento compiuto da Alice e Bob. Infatti, la firma non è distinguibile da quelle usate in un normale schema di firma digitale.

Canali Subliminali in ElGamal

Concretamente, un esempio di canale subliminale può essere realizzato con il metodo di autenticazione di ElGamal esposto nella sezione 2.4.3. Tale implementazione fu proposta dallo stesso Simmons [44].

Supponiamo che Alice voglia mandare un messaggio nascosto M' insieme ad un messaggio innocuo M (entrambi decodificati mediante numeri interi).

- Alice e Bob scelgono di comune accordo un primo grande p ed un generatore g di $\mathbb{Z}_p^\times$.
- Alice sceglie un intero $a \in \{1, \ldots, p-1\}$ e lo condivide con Bob (segretamente). Dopodiché calcola $k_A \equiv g^a \bmod p$ che pubblica.

- Alice calcola
$$r \equiv g^{M'} \bmod p$$
e risolve in v la congruenza
$$M \equiv ar + M'v$$
ovvero,
$$v \equiv (M')^{-1}(M - ar) \bmod p - 1$$
- Alice invia ad Eve la tripla (M, r, v) affinché ne controlli l'autenticità.
- Eve controlla la firma di Alice secondo lo schema di ElGamal, verificando che $(k_A)^r r^v \equiv g^M \bmod p$. Non trovando nulla di strano, invia la tripla a Bob.
- Bob, dopo aver controllato che il messaggio venga da Alice, estrapola il messaggio segreto M' calcolando
$$M' \equiv (M - ar)v^{-1} \bmod p - 1$$

Notiamo che, affinché le congruenze sopra abbiano senso, $\gcd(v, p-1) = 1$ e soprattutto $\gcd(M', p-1) = 1$.

Lo schema proposto è corretto; infatti, siccome $v \equiv (M')^{-1}(M - ar) \bmod p-1$, allora
$$(M - ar)v^{-1} \equiv (M - ar)[(M')^{-1}(M - ar)]^{-1} \equiv M' \bmod p - 1$$

A.2 Funzioni unidirezionali e di Hash

Funzioni unidirezionali

Come introdotto all'inizio della sezione 2.4.2, per poter applicare in concreto la crittografia asimmetrica è necessario disporre di una funzione di cifratura f facile da calcolare ma difficile da invertire. In questo contesto entrano in gioco le *funzioni unidirezionali* che fecero la loro comparsa proprio con la nascita dello schema DH.

Definizione A.2. Una funzione biettiva $f : A \to B$ si dice *funzione unidirezionale* se:

- $f(a)$ è realizzabile con una complessità polinomiale per tutti gli $a \in A$.
- $f^{-1}(b)$ non è realizzabile con una complessità polinomiale per quasi tutti i $b \in B$.
- $f^{-1}(b)$ è realizzabile con una complessità polinomiale se si conoscono altre specifiche tecniche.

Bisogna precisare che con le parole "quasi tutti" si intende che il calcolo di $f^{-1}(b)$ è difficile per tutti i valori di B eccetto alcuni banali. Per esempio, sappiamo che il calcolo del DLP $g^y \equiv x \bmod p$ è computazionalmente complesso, tranne che per $x = 1, -1, g$.

Inoltre, non è stata ancora provata la unidirezionalità di funzioni. Tuttavia, nelle applicazioni vengono usate mappe che si congetturano essere unidirezionali.

Un esempio di tali funzioni è
$$f : \mathbb{Z}_p^\times \to \mathbb{Z}_p^\times \quad a \mapsto g^a \bmod p$$
usata nello scambio di chiavi di Diffie-Hellman 2.4.2. Invertire tale mappa vuol dire risolvere un logaritmo discreto, da cui la presunta unidirezionalità.

Funzioni di Hash

Nelle applicazioni crittografiche a volte è necessario poter accorciare i bit di un messaggio per renderlo più flessibile prima di darlo in pasto ad un protocollo di cifratura. In effetti, un testo in chiaro potrebbe contenere centinaia di migliaia di bit, e per poterlo cifrare occorrerebbe un notevole dispendio di energie e di tempo da parte del computer. Per tale motivo sono state introdotte le funzioni *hash*, dall'inglese "sminuzzare", proprio per fare a fette l'enorme messaggio iniziale. Nel seguito, denoteremo con $\{0,1\}^*$ tutte le possibili combinazioni di "parole" formate dalle sole "lettere" 0 e 1; tuttavia, le funzioni di hash possono essere generalizzate ad alfabeti $\mathcal{A}$ qualunque.

Definizione A.3. Una *funzione di hash* H è una mappa

$$H : \{0,1\}^* \to \{0,1\}^n$$

dove $n \in \mathbb{N}$ è fissato.

Inoltre, $H(x)$ è detta *impronta* di x.

Nelle applicazioni, $n = 128, 224, 256, 384, 512$. Quindi, una tale funzione prende in input una parola di lunghezza arbitraria e ne restituisce una di lunghezza fissata.

Per scopi crittografici, le funzioni di hash devono soddisfare i seguenti requisiti.

- Devono essere funzioni unidirezionali secondo la definizione A.2.
- Devono essere di pubblico dominio, cioè tutti gli utenti di un sistema crittografico devono poterne beneficiare.
- Devono essere *fortemente prive di collisioni*, ovvero, determinare una coppia $(a, b) \in \{0,1\}^* \times \{0,1\}^*$ con $a \neq b$ tale che $H(a) = H(b)$, deve essere un problema computazionalmente intrattabile. In altre parole, devono identificare univocamente il messaggio.

Esempi di utilizzo delle funzioni hash si possono trovare nelle firme digitali come quella di ElGamal. Infatti, invece di firmare un messaggio M lungo milioni di bit, è più conveniente apporre la firma ad $H(M)$ poiché di dimensioni ridotte; il tutto a vantaggio dell'efficienza del sistema.

Analogamente è possibile verificare l'integrità di un messaggio confrontando la sua immagine hash prima e dopo la trasmissione. Infatti, il valore $H(M)$ potrebbe essere intercettato e modificato, ma quando Bob decifra il messaggio riottenendo M, basta che ricalcoli $H(M)$ per accorgersi della manomissione (per le proprietà delle funzioni hash).

Citiamo alcune fra le più importanti ed utilizzate funzioni di hash: la famiglia *SHA-1* e le successive *SHA-2* ed *SHA-3*, utilizzate nei protocolli TLS; *MD5* e l'aggiornamento *MD6*; infine le funzione *BLAKE*.

A.3 Funzioni Pseudocasuali

Una definizione rigorosa di *funzione pseudocasuale (PRF)* potrebbe essere pesante e potrebbe non esternare l'essenza di questi oggetti crittografici. Per tale motivo daremo una definizione più snella e più vicina all'intuizione.

Definizione A.4. Un *oracolo random* è una macchina che, data in input una domanda, la risposta (output) è del tutto casuale, cioè scelta uniformemente all'interno dell'insieme degli output. Inoltre, una volta assegnata una risposta ad una domanda, la risposta a quella domanda sarà sempre la stessa.

Definizione A.5. Sia data $k \in \{0,1\}^n$ detta *chiave* (o *seme*), e sia

$$F_k : \{0,1\}^n \to \{0,1\}^n$$

una funzione. Allora la mappa F_k è detta *funzione pseudocasuale (PRF)* se emula il comportamento di un oracolo random. In particolare, non esiste un algoritmo efficiente in grado di distinguere tra una F_k ed un oracolo random.

Quindi, una funzione pseudocasuale dipende da un parametro segreto k che ne determina l'intera anatomia. La funzione F_k prende in input dei messaggi M e restituisce dei valori che sembrano del tutto casuali, nel senso che non dovrebbe esistere un modello che permetta di scorgere una relazione tra input e output della funzione.

Tuttavia, qualora un malintenzionato riesca ad impossessarsi della chiave k, potrebbe riuscire a costruire la funzione F_k indovinando, così, l'output dato l'input.

Più formalmente, detta R_n la distribuzione uniforme sull'insieme

$$\{f \mid f : \{0,1\}^n \to \{0,1\}^n\}$$

e FR_n la distribuzione sull'insieme

$$\{F_k : \{0,1\}^n \to \{0,1\}^n \mid k \text{ casualmente scelto in } \{0,1\}\ \}$$

allora R_n ed FR_n sono computazionalmente indistinguibili. Ovvero, non esistono algoritmi con complessità polinomiale in grado di distinguere R_n da FR_n.

Solitamente, le funzioni pseudocasuali non vengono mai usate da sole, ma integrate in protocolli crittografici per aggiungere sicurezza al sistema.

Esse possono essere costruire a partire dai *generatori di numeri pseudocasuali (PRNG)* ma non entriamo nei dettagli, poiché questo argomento esula dallo scopo finale di questa tesi.

Chiediamoci, invece, come è possibile costruire una tale funzione, per esempio, nell'ambito delle curve ellittiche. Ovvero, vediamo come si può definire una funzione pseudocasuale

$$H : E(\mathbb{F}_p) \to \mathbb{Z}_q$$

dove E è una curva definita sul campo $\mathbb{F}_p$ con p e $q = \#E$.

Supponiamo che p sia un primo di n bit e consideriamo la mappa

$$f : E(\mathbb{F}_p) \to \{0,1\}^n$$

che dato un punto P sulla curva ellittica, considera la rappresentazione binaria della sua ascissa e la divide a metà, a questi due elementi è successivamente applicata una permutazione. In sostanza, dato un input ci sono due possibili risultati: la coordinata x senza cambiamenti, oppure i primi $n/2$ bit di x sono scambiati con i secondi $n/2$. A

questo punto, possiamo prendere una qualsiasi funzione di hash $h : \{0,1\}^n \to \{0,1\}^n$ e definire

$$H : E(\mathbb{F}_p) \to \{0,1\}^n \quad H(P) = h(f(P))$$

In effetti, siccome i punti di una curva ellittica sono uniformemente distribuiti in $E(\mathbb{F}_p)$, allora lo sono anche le immagini di f in $\{0,1\}^n$. Inoltre, tali immagini, presi come input di h, sono tali che gli output di h sono uniformemente distribuiti in $\{0,1\}^n$. Ciò significa che abbiamo costruito una funzione pseudocasuale H con insieme d'arrivo $\{0,1\}^n$, ciò non significa che sia anche pseudocasuale in $\mathbb{Z}_q$. Fortunatamente, nelle applicazioni concrete $p \approx q$ quindi, l'output di H avrà una distribuzione molto simile a quella uniforme in $\mathbb{Z}_q$.

In conclusione, le funzioni pseudocasuali servono per introdurre casualità all'interno di un algoritmo, nel senso che, dati un input casualmente e uniformemente scelto x e una funzione pseudocasuale h, allora $h(x)$ appare (quasi) uniformemente distribuito all'interno dell'insieme degli output.

Appendice B

Pseudocodici

B.1 SETUP in ECDH

Algoritmo 2: ECDH - Generazione delle chiavi

input : E, q, G, n
output: chiave pubblica A

$x \in \{1, \ldots, n-1\}$
$A \leftarrow [x]G$, pubblicazione A
return A

Algoritmo 3: SETUP in ECDH - Generazione delle chiavi

input : E, q, G, n, Y, a, b, w
output: chiave manomessa A_2

Primo utilizzo del dispositivo:
$x_1 \in \{1, \ldots, n-1\}$
$A_1 \leftarrow [x_1]G$, pubblicazione A_1
x_1 salvato in memoria

Secondo utilizzo del dispositivo:
$t \in \{0, 1\}$
$Z \leftarrow [x_1 - wt]G + [-ax_1 - b]Y$
$x_2 \leftarrow H(Z)$
$A_2 \leftarrow [x_2]G$
return A_2

Algoritmo 4: SETUP in ECDH - Recupero della chiave privata

input : $E, q, G, n, A_1, A_2, a, b, w$
output: chiave privata x_2

$R \leftarrow [a]A_1 + [b]G$
$Z_1 \leftarrow A_1 - [x]R$
if $A_2 = [H(Z_1)]G$ **then**
    return $H(Z_1)$
else
    $Z_2 \leftarrow Z_1 - [w]G$
    return $H(Z_2)$
end

B.2 SETUP in ECEE

Algoritmo 5: ECEE - Cifratura

input : E, q, G, n, M
output: testo cifrato (R, S)

$b \in \{1, \ldots, n-1\}$
$B \leftarrow [b]G$, pubblicazione (E, q, G, B)
$k \in \{1, \ldots, n-1\}$
$R \leftarrow [k]G$
$S \leftarrow M + [k]B$
(R, S) inviata al destinatario
return (R, S)

Algoritmo 6: SETUP in ECEE - Cifratura dei messaggi

input : $E, q, G, n, Y, a, b, w, M_1, M_2$
output: cifratura manomessa (R_2, S_2)

$x_B \in \{1, \ldots, n-1\}$
$B \leftarrow [x_B]G$, pubblicazione (E, q, G, B)
Primo utilizzo del dispositivo:
$k_1 \in \{1, \ldots, n-1\}$
$R_1 \leftarrow [k_1]G$
$S_1 \leftarrow M_1 + [k_1]B$
(R_1, S_1) inviata al destinatario
k_1 salvato in memoria
Secondo utilizzo del dispositivo:
$t \in \{0, 1\}$
$Z \leftarrow [k_1 - wt]G + [-ak_1 - b]Y$
$k_2 \leftarrow H(Z_x)$
$R_2 \leftarrow [k_2]G$
$S_2 \leftarrow M_2 + [k_2]B$
(R_2, S_2) inviata al destinatario
return (R_2, S_2)

Algoritmo 7: SETUP in ECEE - Recupero del messaggio M_2

input : $E, q, G, n, R_1, R_2, S_2, a, b, w$
output: messaggio M_2

$R \leftarrow [a]R_1 + [b]G$
$Z_1 \leftarrow R_1 - [x]R$
if $R_2 = [H(Z_1)]G$ **then**
    $k_2 = H(Z_1)$
else
    $Z_2 \leftarrow Z_1 - [w]G$
    $k_2 = H(Z_2)$
end
$M_2 \leftarrow S_2 - [k_2]B$
return M_2

B.3 SETUP in ECES

Algoritmo 8: ECES - Firma

input : E, q, G, n, f, m
output: firma (R, s) più messaggio m

$a \in \{1, \ldots, n-1\}$
$A \leftarrow [a]G$; pubblicazione (E, q, G, f, A)
$k \in \{1, \ldots, n-1\}$
$R \leftarrow [k]G$
$s \leftarrow k^{-1}(m - af(R)) \bmod n$
(m, R, s) inviata al destinatario
return (m, R, s)

Algoritmo 9: SETUP in ECES - Firma dei messaggi

input : $E, q, G, n, Y, a, b, w, m_1, m_2$
output: firma manomessa (R_2, s_2)

$x_A \in \{1, \ldots, n-1\}$
Primo utilizzo del dispositivo:
$k_1 \in \{1, \ldots, n-1\}$
$R_1 \leftarrow [k_1]G$
$s_1 \leftarrow k_1^{-1}(m_1 - x_A(R_1)_x) \bmod n$
(m_1, R_1, S_1) inviata al destinatario
k_1 salvato in memoria
Secondo utilizzo del dispositivo:
$t \in \{0, 1\}$
$Z \leftarrow [k_1 - wt]G + [-ak_1 - b]Y$
$k_2 \leftarrow H(Z_x)$
$R_2 \leftarrow [k_2]G$
$s_2 \leftarrow k_2^{-1}(m_2 - x_A(R_2)_x) \bmod n$
(m_2, R_2, s_2) inviata al destinatario
return (R_2, s_2)

Algoritmo 10: SETUP in ECES - Recupero della chiave privata

input : $E, q, G, n, R_1, R_2, s_2, a, b, w$
output: chiave privata x_A

$R \leftarrow [a]R_1 + [b]G$
$Z_1 \leftarrow R_1 - [x]R$
if $R_2 = [H(Z_1)]G$ **then**
    $k_2 = H(Z_1)$
else
    $Z_2 \leftarrow Z_1 - [w]G$
    $k_2 = H(Z_2)$
end
$x_A \leftarrow (R_2)_x^{-1}(m_2 - s_2k_2) \bmod n$
return x_A

B.4 SETUP in ECDSA

Algoritmo 11: ECDSA - Firma

input : E, q, G, n, f, h, m
output: firma (R, s) più impronta $h(m)$

1 $a \in \{1, \ldots, n-1\}$
2 $A \leftarrow [a]G$; pubblicazione (E, q, G, f, A)
3 $k \in \{1, \ldots, n-1\}$
4 $e \leftarrow h(m)$
5 $R \leftarrow [k]G$
6 $r \leftarrow f(R) \bmod n$
7 **if** $r \equiv 0 \bmod n$ **then**
8 | ritornare alla riga 3
9 **end**
10 $s \leftarrow k^{-1}(e + ar) \bmod n$
11 **if** $s = 0$ **then**
12 | ritornare alla riga 3
13 **end**
14 (e, R, s) inviata al destinatario
15 **return** (e, R, s)

Algoritmo 12: SETUP in DSA - Firma dei messaggi

input : $E, q, G, n, Y, a, b, w, m_1, m_2$
output: firma manomessa (R_2, s_2)

1 $x_A \in \{1, \ldots, n-1\}$
2 **Primo utilizzo del dispositivo:**
3 $k_1 \in \{1, \ldots, n-1\}$
4 $e_1 \leftarrow h(m_1)$
5 $R_1 \leftarrow [k_1]G$
6 $r_1 \leftarrow (R_1)_x \bmod n$
7 $s_1 \leftarrow k_1^{-1}(e_1 + x_A r_1) \bmod n$
8 (e_1, R_1, s_1) inviata al destinatario
9 k_1 salvato in memoria
10 **Secondo utilizzo del dispositivo:**
11 $t \in \{0, 1\}$
12 $Z \leftarrow [k_1 - wt]G + [-ak_1 - b]Y$
13 $k_2 \leftarrow H(Z_x)$
14 $e_2 \leftarrow h(m_2)$
15 $R_2 \leftarrow [k_2]G$
16 $r_2 \leftarrow (R_2)_x \bmod n$
17 $s_2 \leftarrow k_2^{-1}(e_2 + x_A r_2) \bmod n$
18 (e_2, R_2, s_2) inviata al destinatario
19 **return** (R_2, s_2)

Algoritmo 13: SETUP in DSA - Recupero della chiave privata

input : $E, q, G, n, R_1, R_2, s_2, a, b, w$
output: chiave privata x_A

$R \leftarrow [a]R_1 + [b]G$
$Z_1 \leftarrow R_1 - [x]R$
if $R_2 = [H(Z_1)]G$ **then**
    $k_2 = H(Z_1)$
else
    $Z_2 \leftarrow Z_1 - [w]G$
    $k_2 = H(Z_2)$
end
$x_A \leftarrow r_2^{-1}(s_2k_2 - e_2) \bmod n$
return x_A

Lista dei Simboli

DLP	Problema del Logaritmo Discreto (Discrete Logarithm Problem)
DH	Protocollo di Diffie-Hellman (Diffie-Hellman Keys Exchange)
DHP	Problema di Diffie-Hellman (Diffie-Hellman Problem)
DSA	Digital Signature Algorithm
ECC	Crittosistema su Curve Ellittiche (Elliptic Curve Cryptosystem)
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDHP	Elliptic Curve Diffie-Hellman Problem
ECDH	Elliptic Curve Diffie-Hellman Keys Exchange
ECEE	Elliptic Curve ElGaml Encryption
ECES	Elliptic Curve ElGamal Signaure
ECDSA	Elliptic Curve Digital Signature Algorithm
SETUP	Secretly Embedded Trapdoor with Universal Protection
DLK	Discrete Log Kleptogram
ECDLK	Elliptic Curve Discrete Log Kleptogram

Bibliografia

[1] Sergei Abramovich e Yakov Nikitin. «On the probability of co-primality of two natural numbers chosen at random (Who was the first to pose and solve this problem?)» In: (ago. 2016).

[2] M. Antheunisse. «Kleptography: cryptography with backdoors». Tesi di laurea mag. Eindhoven University of Technology, Department of Mathematics e Computer Science, 2015.

[3] Paul Benioff. «Quantum mechanical Hamiltonian models of Turing machines». In: *Journal of Statistical Physics* 29 (nov. 1982), pp. 515–546. DOI: 10.1007/BF01342185.

[4] Daniel Bernstein. «Curve25519: New Diffie-Hellman Speed Records». In: vol. 3958. Apr. 2006, pp. 207–228. DOI: 10.1007/11745853_14.

[5] Daniel J. Bernstein e Tanja Lange. *SafeCurves: choosing safe curves for elliptic-curve cryptography*. 2014. URL: https://safecurves.cr.yp.to/.

[6] Daniel Bernstein e Tanja Lange. «Inverted Edwards Coordinates.» In: vol. 2007. Gen. 2007, pp. 20–27.

[7] Ian F. Blake, G. Seroussi e N. P. Smart. *Elliptic Curves in Cryptography*. USA: Cambridge University Press, 1999. ISBN: 0521653746.

[8] Maurizio Cailotto. *Curve Algebriche Piane*. 2012. URL: https://www.math.unipd.it/~maurizio/cap/CAP2012.pdf.

[9] Ran Canetti, John Friedl e Igor Shparlinski. «On Certain Exponential Sums And The Distribution Of Diffie-Hellman Triples». In: *Journal of the London Mathematical Society* 59 (mag. 2000). DOI: 10.1112/S002461079900736X.

[10] Ran Canetti, John Friedlander, Sergei Konyagin, Michael Larsen, Daniel Lieman e Igor Shparlinski. «On The Statistical Properties Of Diffie-Hellman Distributions». In: *Israel Journal of Mathematics* 120 (dic. 1998). DOI: 10.1007/s11856-000-1270-1.

[11] U.S. Departemente of Commerce/National Institute of Standards e Technology. *Recommendations for Discret Logarithm-Based Cryptography: Elliptic Curve Domain Parameters*. 2019. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-186-draft.pdf.

[12] *Computational complexity of mathematical operations*. URL: https://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations.

[13] D. Coppersmith. «Fast evaluation of logarithms in fields of characteristic two». In: *IEEE Transactions on Information Theory* 30.4 (1984), pp. 587–594.

[14] W. Diffie e M. Hellman. «New directions in cryptography». In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654.

[15] *Discrete Logarithm Records*. URL: https://en.wikipedia.org/wiki/Discrete_logarithm_records.

[16] Harold Edwards. «A normal form for elliptic curves». In: *Bulletin of The American Mathematical Society - BULL AMER MATH SOC* 44 (lug. 2007), pp. 393–423. DOI: 10.1090/S0273-0979-07-01153-6.

[17] Taher ElGamal. «A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms». In: (1985). A cura di George Robert Blakley e David Chaum, pp. 10–18.

[18] Steven D. Galbraith e Pierrick Gaudry. «Recent progress on the elliptic curve discrete logarithm problem». In: (2015). https://eprint.iacr.org/2015/1022.

[19] Neal Koblitz. *A Course in Number Theory and Cryptography*. Springer, 1994.

[20] Neal Koblitz. «Elliptic curve cryptosystem». In: *Mathematics of Computations* (1987).

[21] Bohdan Kovalenko e Anton Kudin. *Evaluation and minimization of kleptography risks in cryptographic algorithms*. 2020. URL: https://web.math.pmf.unizg.hr/cecc2020/files/kovalenko-kudin-talk.pdf.

[22] Bohdan Kovalenko e Anton Kudin. *Kleptography trapdoor free cryptographic protocols*. Cryptology ePrint Archive, Report 2018/989. 2018. URL: https://eprint.iacr.org/2018/989.

[23] Daniel Kucner e Miroslaw Kutylowski. «Stochastic kleptography detection». In: *Public-Key Cryptography and Computational Number Theory*. Berlin, Boston: De Gruyter, 31 Dec. 2001, pp. 137–150. ISBN: 9783110170467. DOI: https://doi.org/10.1515/9783110881035.137. URL: https://www.degruyter.com/view/book/9783110881035/10.1515/9783110881035.137.xml.

[24] Alessandro Languasco e Alessandro Zaccagnini. *Manuale di Crittografia. Teoria Algoritmi e protocolli*. Hoepli, 2015.

[25] D.A. Marcus. *Number Fields*. Universitext. Springer New York, 2012. ISBN: 9781468493566.

[26] Victor S. Miller. «Use of Elliptic Curves in Cryptography». In: *Advances in Cryptology*. CRYPTO '85. Berlin, Heidelberg: Springer-Verlag, 1985, pp. 417–426. ISBN: 3540164634.

[27] Elsayed Mohamed e Elkamchouchi Hassan. «Elliptic Curve Kleptography». In: *International Journal of Computer Science and Network Security* 10 (2010), pp. 183–185.

[28] Elsayed Mohamed e Elkamchouchi Hassan. «Kleptographic Attacks on Elliptic Curve Cryptosystems». In: *International Journal of Computer Science and Network Security* 10 (2010), pp. 213–215.

[29] Elsayed Mohamed e Elkamchouchi Hassan. «Kleptographic Attacks on Elliptic Curve Signature». In: *International Journal of Computer Science and Network Security* 10 (2010), pp. 264–267.

[30] Peter Montgomery. «Montgomery, P.L.: Speeding the Pollard and Elliptic Curve Methods of Factorization. Math. Comp. 48, 243-264». In: *Mathematics of Computation - Math. Comput.* 48 (gen. 1987), pp. 243–243. DOI: 10.1090/S0025-5718-1987-0866113-7.

[31] NIST. *FIPS PUB 186-4: Digital Signature Standard (DSS)*. 2013. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

[32] Elisabeth Oswald. «Introduction to Elliptic Curve and Cryptography». In: (2002).

[33] Christof Paar e Jan Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. Gen. 2010. ISBN: 978-3-642-04100-6.

[34] Nicole Perlroth. *Government Announces Steps to Restore Confidence on Encryption Standards*. 2010. URL: https://bits.blogs.nytimes.com/2013/09/10/government-announces-steps-to-restore-confidence-on-encryption-standards/.

[35] John Proos e Christof Zalka. «Shor's Discrete Logarithm Quantum Algorithm for Elliptic Curves». In: *Quantum Information & Computation* 3 (feb. 2003).

[36] Certicom Research. *Standards for Efficient Cryptography SEC 2: Recommended Elliptic Curve Domain Parameters*. Ver. 2. 2010. URL: https://www.secg.org/sec2-v2.pdf.

[37] R. L. Rivest, A. Shamir e L. Adleman. «A Method for Obtaining Digital Signatures and Public-Key Cryptosystems». In: *Commun. ACM* 21.2 (feb. 1978), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: https://doi.org/10.1145/359340.359342.

[38] A. Sajjad, M. Afzal, M. M. W. Iqbal, H. Abbas, R. Latif e R. A. Raza. «Kleptographic Attack on Elliptic Curve Based Cryptographic Protocols». In: *IEEE Access* 8 (2020), pp. 139903–139917.

[39] René Schoof. «Elliptic Curves Over Finite Fields and the Computation of Square Roots mod p». In: *Mathematics of Computation* 44.170 (1985), pp. 483–494. ISSN: 00255718, 10886842. URL: http://www.jstor.org/stable/2007968.

[40] Peter Shor. «Algorithms for Quantum Computation: Discrete Logarithms and Factoring». In: *Proceedings of 35th Annual Symposium on Foundations of Computer Science* (ott. 1996). DOI: 10.1109/SFCS.1994.365700.

[41] Peter Shor. «Polynomial-Time Algorithms For Prime Factorization And Discrete Logarithms On A Quantum Computer». In: *SIAM Review* 41 (mag. 1997). DOI: 10.1137/S0036144598347011.

[42] Dan Shumow e Niels Ferguson. *On the Possibility of a Back Door in the NIST SP800-90 Dual Ec Prng*. 2007. URL: https://rump2007.cr.yp.to/15-shumow.pdf.

[43] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*. Second Edition. Springer, 2009.

[44] Gustavus Simmons. «The Subliminal Channel and Digital Signatures». In: mar. 2007, pp. 364–378. DOI: 10.1007/3-540-39757-4_25.

[45] Gustavus J. Simmons. «The Prisoners' Problem and the Subliminal Channel». In: *Advances in Cryptology: Proceedings of CRYPTO '83*. Plenum, 1984, pp. 51–67.

[46] *The Fundamental Theorem of Finite Abelian Groups*. URL: https://demonstrations.wolfram.com/TheFundamentalTheoremOfFiniteAbelianGroups/.

[47] *Things that use Curve25519*. URL: https://ianix.com/pub/curve25519-deployment.html.

[48] S. Vanstone. «Elliptic curve cryptosystem - The answer to strong, fast public-key cryptography for securing constrained environments». In: *Inf. Secur. Tech. Rep.* 2 (1997), pp. 78–87.

[49] Lawrence C. Washington. *Elliptic curves. Number theory and cryptography*. Chapman & Hall/CRC, 2008.

[50] WhatsApp. *WhatsApp Encryption Overview*. 2017. URL: https://scontent.whatsapp.net/v/t39.8562-34/89275998_627986927772871_4167828889579552768_n.pdf?_nc_sid=2fbf2a&_nc_ohc=QwqJ8rYOApgAX8rCsZM&_nc_ht=scontent.whatsapp.net&oh=20652ccbd0faf75edb31c9495a58f654&oe=5FA02451.

[51] Adam Young e Moti Yung. «A Space Efficient Backdoor in RSA and Its Applications». In: *Proceedings of the 12th International Conference on Selected Areas in Cryptography*. SAC'05. Kingston, ON, Canada: Springer-Verlag, 2005, pp. 128–143. ISBN: 3540331085. DOI: 10.1007/11693383_9. URL: https://doi.org/10.1007/11693383_9.

[52] Adam Young e Moti Yung. «A Timing-Resistant Elliptic Curve Backdoor in RSA». In: vol. 4990. Ago. 2007, pp. 427–441. DOI: 10.1007/978-3-540-79499-8_33.

[53] Adam Young e Moti Yung. «Kleptography: Using Cryptography Against Cryptography». In: *Advances in Cryptology — EUROCRYPT '97*. A cura di Walter Fumy. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 62–74. ISBN: 978-3-540-69053-5.

[54] Adam Young e Moti Yung. *Malicious Cryptography: Exposing Cryptovirology*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2004. ISBN: 0764549758.

[55] Adam Young e Moti Yung. «Malicious Cryptography: Kleptographic Aspects». In: *Proceedings of the 2005 International Conference on Topics in Cryptology*. CT-RSA'05. San Francisco, CA: Springer-Verlag, 2005, pp. 7–18. ISBN: 3540243992. DOI: 10.1007/978-3-540-30574-3_2. URL: https://doi.org/10.1007/978-3-540-30574-3_2.

[56] Adam Young e Moti Yung. «The Dark Side of "Black-Box" Cryptography, or: Should We Trust Capstone?» In: *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '96. Berlin, Heidelberg: Springer-Verlag, 1996, pp. 89–103. ISBN: 3540615121.

[57] Adam Young e Moti Yung. «The Prevalence of Kleptographic Attacks on Discrete-Log Based Cryptosystems». In: *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '97. Berlin, Heidelberg: Springer-Verlag, 1997, pp. 264–276. ISBN: 3540633847.

[58] B. David Zarley. «America's first exascale supercomputer to be built by 2021». In: *The Verge* (2019). URL: https://www.theverge.com/2019/3/18/18271328/supercomputer-build-date-exascale-intel-argonne-national-laboratory-energy.