Università degli studi di Modena e Reggio Emilia

Dipartimento di Ingegneria Enzo Ferrari

Crittografia Applicata

Anno Accademico 2023/24

Indice

Capitolo 1

Introduzione

Crittologia: l'arte delle scritture segrete, può essere divisa in 3 macro argomenti:

1. **crittografia**: come *trasformare* messaggi per proteggerne il contenuto (l'informazione).
2. **steganografia**: come *nascondere* messaggi per evitare che venga individuato (ex. **least significant bit steganography**).
3. **crittoanalisi**: come *analizzare* messaggi e rivelarne l'informazione.

1.1 Crittografia Classica

La sicurezza della crittografia classica si basa unicamente sulla **segretezza** del **metodo** (noto solo al *sender* e al *receiver*), considerava come una tipologia di attacco quello **passivo** (*read only*). Basandosi su questi concetti il suo utilizzo in applicazioni reali è molto limitato (nel senso moderno), considerando la comunicazione in termini di scambio di informazioni in linguaggio naturale.
Alcuni esempi: **scytale** (*transposition cipher*), **caesar cipher** (*shift cipher*) e **vigenere cipher**.

1.2 Crittografia Moderna

1.2.1 Encryption only

La crittografia moderna si basa su due principi:

1. **Kerckhoffs principles**:
   - gli algoritmi devono essere pubblici.
   - la sicurezza del metodo si deve basare sulla **segretezza** della **chiave**.
   - uno schema deve essere "praticamente", se non "matematicamente" indecifrabile
2. **Shannon principles**:
   - **confusione**: ogni bit del crittogramma deve dipendere da più bit della chiave, oscurando, però, la correlazione tra le due.
   - **diffusione**: se viene cambiato un singolo bit del testo in chiaro, allora almeno la metà dei bit del crittogramma devono cambiare, e viceversa.

Nella crittografia moderna lo spazio delle chiavi deve essere sufficentemente ampio per evitare una ricerca esaustiva su di esso, in più, nessuna informazione (nè del *plaintext*, né della *key*) deve poter essere estrapolata dal *ciphertext*. Viene detto che il *ciphertext* deve essere **indistinguibile** da una sequenza di bit *random*.

1.2.2 Extended and applied settings

Gli avversari (i crittoanalisti) non sono più unicamente **passivi**, ma bisogna modellare delle tipologia di avversari che siano capaci anche di **interagire** con i nostri sistemi e **manipolare** dei messaggi.
Per ognuna di queste modellazioni è necessario **provare la sicurezza** dei sistemi andando a **definire** delle attività (tramite la comprensione e modellare cosa è "sicuro") e **costruendo** attività (progettandole e provandone la veridicità).

Bisogna definire le **primitive**, gli **schemi**, i **protocolli** e le *applicazioni* che vengono utilizzati, andandoli ad analizzare separatamente e completa.
⇒ può essere necessario costruire schemi e protocolli modellati su misura per applicazioni reali inerenti ad un certo caso d'uso: **Applied Cryptography**.

1.3 Crittografia Applicata

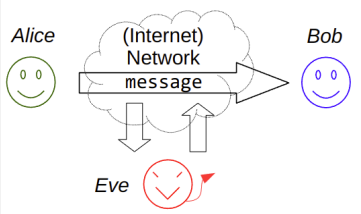
È un layer di astrazione che può essere (quasi) direttamente mappato all'interno di una soluzione per un caso d'uso reale (quindi tecniche "pratiche"). Siccome analizziamo **soluzioni pratiche** bisogna gestire possibili errori dovuti ad **implementazioni** o ***deployment*** errati. I protcolli sicuri assumono che un attaccante tenti di accedere alle informazioni in transito (violazione della **confidenzialità**) e cerchi di impersonificare un mittente (violazione dell'**autenticazione**).
Una delle sicurezze che deve garantire un protocollo sicuro è la **confidenzialità**.

Quando si studi/analizza/progetta un protocollo crittografico è necessario identificare:

- **system model**: descrive lo scenario ("idealmente") di utilizzo, andando a definire: gli attori **legittimi**, la **tipologia di protocollo** utilizzata, le **informazioni** possedute dagli attori legittimi, e altre informazioni sullo scenario applicativo.

  Esempio

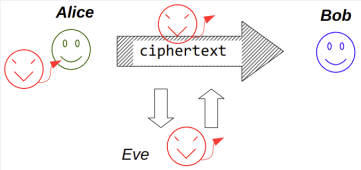
  Un protocollo sicuro che ha come **scopo** proteggere le informazioni scambiati tra due attori: **Alice** e **Bob** quando un possibile attaccante **Eve** può accedere direttamente all'informazione tramite il canale fisico.

- **threat modelling** (modellazione della tipologia di attaccante): abbiamo identificato degli attori legittimi, ma quanto sono affidabili? Modelliamo il protocollo sulla base dell'attaccante.
  1. che operazioni può effettuare sui dati: solo lettura, modifica, inserimento o eliminazione dei dati.
  2. qual è la superficie di attacco e cosa può provare a fare: ha accesso ad alcune funzionalità (cifrazione/decifrazione), che tipologia conoscenza (*white/gray/black box*), quanti tentavi si hanno: adattivo o meno.

  Esempio

  Cosa può fare **Eve** per compromettere la comunicazione: leggere, manipolare le informazioni in transito, compromissione di un attore legittimo.

- **security guarantees**: quale aspetto di sicurezza vogliamo garantire: **confidenzialità**, **integrità** (autenticazione), **disponibilità**, **non ripudio** (è anche presente il concetto di ***forward security***).
- **cryptogrphy settings**: le due classi principali sono: crittografia **simmetrica** (le funzioni di *encrypt* e *decrypt* utilizzano lo stesso **segreto**) e crittografia **asimmetrica** (sono presenti due differenti **chiavi**, uno utilizzabile durante la funzione di *encrypt* - *public* - e l'altro utilizzato durante la funzione di *decrypt* - *secret*).
- **security assumptions of a proposed scheme**

Alcuni **protocolli**:

1. ***Secure key exchange protocol*** (scambio sicuro di chiavi): Alice e Bob non hanno nessuna **chiave**, ne vogliono ottenere una **sicura** e **condivisa** comunicando su un canale sincrono e non sicuro.
2. ***Secure storage***: gli algoritmi di crittografia possono aggiungere protezione su dati conservati in canali protetti, l'avversario ha avuto accesso ai dati dopo aver sconfitto le difese iniziali, negli scenari di storage possiamo andare ad analizzare due tipologie di dati: "*data at rest*" (è lo standard e la *best-practice*) oppure "*data in use*" (continua ricerca soprattuto per i dispositivi mobili).
3. ***(Identity) Authentication***: (***challenge-response protocols***) ovvero dimostrare il possesso di un segreto senza mandarlo.
4. ***Password Protection***: gli schemi crittografici possono "proteggere" le password in caso di *data breaches*, non solo usando l'hash (anche se aggiunto il *salt*), ma anche tecniche per prevenire il ***brute-forcing*** attraverso **ASICs** (**Application-specific integrated circuit**).

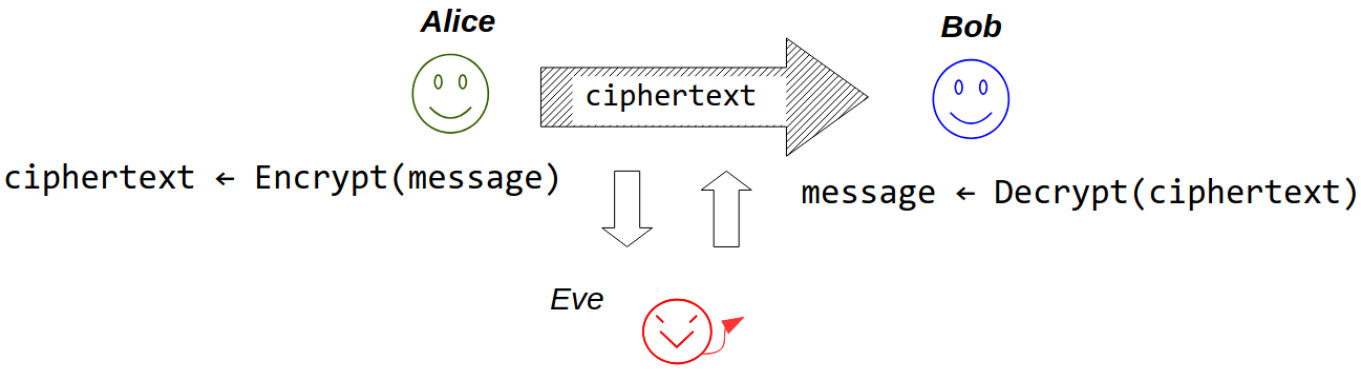
Per riassumere quanto visto fino ad adesso, possiamo dire che la **crittografia applicata** è l'insieme di molti ***system models*** (comunicazioni sincroni, messaggi di gruppi asincroni, *disk encryption*, ...), molti ***security guarantees*** (*information security*), integrità e autenticazione, molti **type of attackers** (passivi e attivi oppure online e offline). La **crittografia applicata** mira a dimostrare la sicurezza sfruttando "strati inferiori":

- la sicurezza dei **protocolli** viene ridotta alla sicurezza sugli schemi.
- la sicurezza degli **schemi** viene ridotta alla sicurezza delle primitive.
- la sicurezza delle **primitive** viene ridotta alla robusteza di problemi matematici.

Per progettare un **protocollo di comunicazione sicuro** è necessario utilizzare un approccio modulare basato su uno *stack* di altri "oggetti", un approccio tipico per lo *stack* è quello che ogni *layer* ha un determinato compito: il protocollo o lo schema è modificato su certi ***cryptographic settings*** e la sicurezza viene provato contro uno specifico tipo di avversario (***security models***).

Capitolo 2

Crittografia Simmetrica



Le garanzie di sicurezze che si cercano di mantenere sono:

- **confidenzialità**: Eve non può accedere a nessuna della informazioni sul messaggio.
- **autenticazione**: Bob può verificare se il messaggio non è stato inviato da Alice, viene anche chiamata ***data orgin authenticity*** nel contesto della comunicazioni e implica anche la protezione contro modifiche illegittime (**integrità**).

La sicurezza non esiste in natura è quindi necessario idearla e modellarla, questa prima parte prende il nome di *Definitional Activity*. È comunque importante ricordare che le **definizioni** possono essere **errate** principalmente per errori nella modellazione o nello sviluppo software, ma anche perché non si è stato in grado di modellare quello che era invece richiesto. Un altro errore che si può essere portati a fare è quello di utilizzare in maniera errata certe definizioni ad esempio al di fuori del contesto per cui era stata definita.

Definitional Activity: permette di descrivere che cosa l'avversario può **fare** e cosa può **vedere**. Esistono molteplici modi per definire la sicurezza in maniera più formale, uno tra questi è la ***simulation-based security*** dove viene definita una funzione ideale che soddisfa la definizione di security e poi dimostrare che la funzione costruita si comporti come quella ideale.

First Adversary Model

Quindi modelliamo e identifichiamo le casistiche e tipologie di un attaccante.

Attack Model: Passive Eavesdropper (EAV)

Ha capacità di lettura dei soli *ciphertext* e non è capace di **scegliere nulla**

Security Goal: Indistinguishably

L'avversario non può distinguere il *ciphertext* da sequenza di caratteri random.

Modellando in questo modo il nostro avversario è possibile osservare che non viene descritto nulla sul nascondere la lunghezza del *plaintext*, infatti per questa prima modellazione l'avversario può vedere la lunghezza del *plaintext*.
Nota: la crittografia non ha come obiettivo quello di nascondere la lunghezza del testo in chiaro, nel caso in cui questa informazione fosse confidenziale, è necessario proteggerla a livello applicativo.

Le capacità di un avversario vengono espresse e descritte tramite degli algoritmi chiamati **esperimenti**, che vengono eseguiti da un'entità chiamata ***challenger*** (che per semplicità andiamo ad identificare nell'attore onesto).
Come prima andiamo ad analizzare ***IND-EAV***, il *challenger* va a scegliere un messaggio **m** che viene scelto con la stessa probabilità tra:

- dati random: $m \leftarrow \{0,1\}^n$
- un messaggio generato attraverso la cifrazione $m = \text{Encryption}(p)$, dove **p** può essere scelto nello stesso modo di prima $p \leftarrow \{0,1\}^n$

All'avversario viene fornito **m** e deve scegliere se è un messaggio randomico o se è l'output dell'*encryption*, l'avversario vince l'esperimento se la sua decisione è corretta.

IND-EAV: Perfect & Computational Indistinguishably

Andremo a discutere due tipologie di sicurezza:

1. ***perfect***: la probabilità dell'avversario di vincere l'esperimento è del **50%**, viene anche chiamata ***Unconditional Security*** o ***Informatition Theoretic Security***.
2. ***computational***: la probabilità dell'avversario di vincere l'esperimento è **50%** più una **quantità trascurabile**.

Qualunque tipologia di schema **praticabile** garantisce **sicurezza computazionale**, e se capace di essere sicuro contro un'**esperimento IND-EAV** viene detto ***IND-EAV secure***.

2.1 Sicurezza Incondizionata & One-Time Pad

XOR: gli schemi di crittografia moderni sono progettati per **dati binari**. L'operazione base per la crittografia simmetrica è lo **XOR**.

$$c = m \oplus k$$

m	k	c
0	0	0
0	1	1
1	0	1
1	1	0

Nota: lo **XOR** può essere anche modellato come la somma bit per bit modulo 2: $c_i = (m_i \oplus k_i) \bmod 2$

Lo XOR viene scelto perché dato un certo **m**, se **k** viene scelta in maniera randomica la probabilità di **c** di essere **0** o **1** è $\mathbf{p = 0.5}$.
In questo modo sapere **c** non da informazioni su **m** e quindi **c** è indistinguibile da una successioni di bit random: $\{0,1\}^n$

One-Time Pad - Vernam's Cipher: è un algoritmo di crittografia che esegue un XOR bit a bit tra il testo in chiaro e la chiave, le due lunghezze devono essere uguali e la chiave devere essere random. $c_i = m_i \oplus k_i \; \forall i \in \{0, ..., n\}$ dove n è la lunghezza del testo in chiaro.
Per la decifrazione bisogna utilizzare la stessa chiave: $m = c \oplus k = (m \oplus k) \oplus k = m \oplus (k \oplus k) = m$.
Anche se **OTP** è **incondizionatamente sicuro** non è praticabile realmente in quanto la generazione della chiave per testo arbitrario è computazionalmente onerosa ed è un algoritmo completamente **malleabile**. Gli schemi di crittografia oggi usati sono **computazionalmente sicuri**.

Nota sulla randomicità in crittografia: la randomicità in crittografia è differente da quella "statistica", ovvero una **distribuzione uniforme di 0 e 1** (che è necessaria ma non sufficiente), ma deve essere ***unpredictable***, in modo tale che anche osservando una sequenza, più o meno lunga di bit, non sia possibile predirre il bit successivo.

2.2 Sicurezza Computazionale: Security Level e Key Sizes

Gli schemi crittografici moderni hanno come parte dei requisiti i ***Kerckhoffs principle*** e necessitano uno **spazio delle chiavi largo** abbastanza per prevenire attacchi di ricerca esaustiva. Inoltre lo schema deve essere progettato in modo che si possa prevenire crittanalisi sul crittogramma, quindi nessuna informazione deve essere ottenuta dal crittogramma indipendentemente dal tipo di dato e deve essere sicura contro l'esperimento IND-EAV.

Le condizioni necessarie avere degli schemi computazionalmente sicuri sono:

- gli schemi utilizzati devono essere computazionalmente sicuri.
- definiamo F_k come una **PRF - *Pseudo-Random Function*** con una chiave fissa **k** scelta randomicamente.
  - la **chiave** deve essere "**corta**" (ma lunga abbastanza per resistere ad attacchi a forza bruta).
  - deve essere capace di cifrare grandi moli di dati.
  - data la **chiave** le funzioni di ***encryption*** e ***decryption*** devono essere **efficenti**.
  - senza la **chiave** la probabilità di rompere lo schema crittografico deve essere **trascurabile**.

È necessario tradurre in termini algoritmici **efficenti** e **trascurabile**. Alice e Bob che usano la funzione di *encryption* e *decryption* con la chiave devono essere capaci di eseguire gli algoritmi con costo *efficient*, quindi il **costo computazionale** e **di memorizzazione** sono **polinomiali** sui parametri di sicurezza. Eve, che non conosce la chiave deve operare in maniera attraverso algoritmi **inefficienti**.
→ se il costo dell'attacco diverge da quello degli attori legittimi, è possibile scegliere i parametri di sicurezza appropriati in modo tale che la probabilità di completare correttamente l'algoritmo si molto piccola: **trascurabile**.

Se fissiamo come probabilità di successo per definire un attacco a *brute force* **inefficiente** 10^{-6}, identifichiamo il valore di **N** per funzioni che hanno costo computazionale diverso, per quali valori di $\mathbf{n} > \mathbf{N}$ le probabilità di sucesso sono inferiori?

Costo di Esecuzione		**Probabilità di Successo**		***Threshold*, b = 2**
$\mathbf{O}(b^n)$	→	$\mathbf{O}(b^{-n})$	→	**N = 20**
$\mathbf{O}(b^{\sqrt{n}})$	→	$\mathbf{O}(b^{-\sqrt{n}})$	→	**N = 400**
$\mathbf{O}(b^{\log n})$	→	$\mathbf{O}(b^{-\log n})$	→	**N = 32**

La conoscenza del costo dell'attacco più noto determina il valore del parametro di sicurezza, tra gli altri, la **dimensione della chiave**, identificato dal valore **N**.

$$\exists N \mid f(n) < \tfrac{1}{p(n)},\ \forall n < N$$

Esempio

Definiamo il costo di cifrazione $c_{enc}(n) = n$ mentre il costo dell'attacco $c_{attack} = n^2$ dove n è la lunghezza della chiave.

Negli anni 2000 l'*encryption* utilizzava una chiave a 64bit e impiegava 1ms, mentre l'attacco a forza bruta, impiegava 2 anni. Dopo 10 anni, nel 2010, con la stessa chiave la cifrazione impiegava 0.1ms e il suo brute force 2 mesi.

Aumentando la lunghezza della chiave, raddoppiandola, la fase di cifrazione impiegava 0.2ms, mentre quella di *brute force* passava da 2^{64} a $2^{128} \simeq 10^{20}$ mesi.

Grazie a nuove scoperte vengono trovati algoritmi che **indeboliscono** o **compromettono** il cifrario. Ad esempio alcuni schemi vengono pubblicamente violati pochi anni dopo la loro scoperta come gli schemi crittografici della famiglia *rc* o *sha1*. È anche possibile che schemi standard vengano indeboliti attraverso *backdoor*, parametri deboli o "particolari" e implementazioni deboli.

Efficient function* → *polynomial

Il costo (computazionale e memorizzativo) sono polinomiali rispetto ad un certo parametro di sicurezza n, algoritmi di *encryption* costano al massimo:

$$p(n) := a \cdot n^x$$

Negligible function* → *smaller that any inverse polynomial

Esiste un valore di **N** tale che la funzione sia minore di qualsialsi funzione polinomiale:

$$\exists N \text{ t.c. } f(n) < \tfrac{1}{p(n)},\ \forall\ n < N$$

PseudoRandom functions

Definiamo una **funzione ideale** che soddisfa computazionalmente l'esperimento di sicurezza IND-EAV, nel caso di crittografia a chiave privata, questo tipo di funzione si chiama ***(keyed) family of PseudoRandom Function (PRF)***.

$$\mathbf{F} \ : \ \mathbf{K} \times \mathbf{P} \mapsto \mathbf{C}$$

Dove:

- **K** è uniformemente scelto da $\{0,1\}^{Lk(n)}$
- **P** è il *plaintext* scelto arbitrariamente da $\{0,1\}^{Lp(n)}$
- **C** soddisfa computazionalmente **IND-EAV**, dove la "quantità trascurabile" è espressa dalla funzione **negl(n)**

Uno schema di crittografia deve essere **funzionale**. Definiamo **F** come la **PRF** allora **F** si definirà computazionalmente sicura se:

- lo spazio della chiavi è "**piccolo**", ma grande a sufficienza per resistere ad attacchi basati su ricerca esaustiva. Quindi $Lk(n)$ **deve** essere una funzione efficiente.
- ha la capacità di generare in output grande quantità di dati ***pseudorandom*** (è sicuro per IND-EAV).
- il costo di computazione di **F** è **efficiente**.
- senza la chiave, la probabilità di rompere lo schema crittografico è **trascurabile**, il costo di calcolare $\mathbf{F}^{-1}$ è **inefficiente**.

Concrete parameters for acceptable security guarantees

Gli schemi di crittografia (simmetrica) moderni vengono considerati computazionalmente sicuri, tali schema possono essere violati se si dispone di abbastanza tempo e abbastanza risorse.
Il ***Security Level*** dello schema è la media del numero di operazioni necessarie per rompere lo schema: gli standard stabiliscono dei valori tali che la quantità di tempo e risorse necessaria per calcolare tale quantità di operazioni è *unfeasible*.

- 80-bit di sicurezza $\rightarrow$ 2^{80} operazioni in media per rompere lo schema (insicuro dal 2010).
- 112-bit di sicurezza $\rightarrow$ 2^{112} operazioni in media per rompere lo schema (insicuro dal 2030).
- 128-bit di sicurezza $\rightarrow$ 2^{128} operazioni in media per rompere lo schema (stimata la sicurezza per ogni scenario successivo).

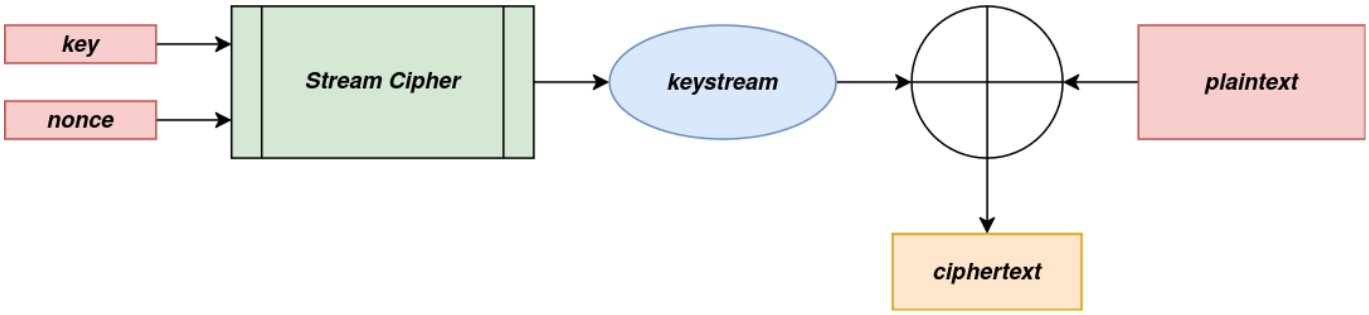
Nei moderni **schemi di crittografia simmetrica**, la **lunghezza della chiave** definisce il **livello di sicurezza**, in quanto l'attacco *best-known* è basato sull'indovinare il segreto. A differenza, negli **schemi di crittografia asimmetrica** dove invece è presente solo una correlazione, in quanto dipende dagli attacchi noti alla matematica sottostante.
Software e librerie **dovrebbero** implementare configurazioni **sicure** di **default** e aggiornate se necessarie

Asymmetric Cryptography & Quantum Computers (PQC)

Si stima che gli attuali standard di crittografia asimmetrica saranno efficacemente violati dai computer quantistici nei prossimi decenni. Nell'ultimo decennio, sono stati ipotizzati e analizzati a fondo nuovi problemi cosiddetti "***post-quantum hard problems***", ovvero che non possono essere risolti in modo efficiente nemmeno con un computer quantistico.

2.3 Stream Cipher

Il primo approccio per implementare una funzione "reale" che approssimi la funzione ideale PRF. Gli *stream cipher* sono ***Deterministic Pseudorandom Bitstring Generators (DPBG)***. Gli schemi a flusso sono funzioni **deterministiche** che prendono in input ***small random input*** e generano come output dati che non possono essere distinti (***indistinguishably***) da dati random e non su cui non si può fare una previsione (***unpredictable***), il dato **pseudo-random** viene chiamato ***keystream***, la funzione di cifrazione richiede xor*are* il *keystream* con il messaggio, infatti gli *stream cipher* cercano di approssimare il **OTP**



Il ***nonce*** è un valore non confidenziale che **deve** essere **univoco** per ogni operazioni di *encryption*. Uno dei più popolari cifrari a flusso è ***ChaCha20***, utilizzo:

```
1    $ echo Hello! | openssl chacha20 -pbkdf2 -k pippo -a -e | \
2        openssl chacha20 -pbkdf2 -k pippo -a -d
```

Questa tipologia di schema è **malleabile** ed è vulnerabile a **riutilizzo della chiave**, sia in caso di messaggi diversi che nel caso di due stati diversi di un certo file (dipende dal contesto)

$$c_1 = m_1 \oplus \text{DPGB}(k) \qquad c_2 = m_2 \oplus \text{DPGB}(k)$$

$$c_1 \oplus c_2 = (m_1 \oplus \text{DPGB}(k)) \oplus (m_2 \oplus \text{DPGB}(k)) = (m_1 \oplus m_2) \oplus (\text{DPGB}(k) \oplus \text{DPGB}(k)) = m_1 \oplus m_2$$

In questo modo la *keystream* viene generata in modo che dipenda unicamente dalla chiave, se noi andiamo a ad utilizzare un altro valore (***nonce***) è possibile rimuovere la vulnerabilità di riutilizzo della chiave. Definendo k_i la *keystream* nell'istante i-esimo avremo:

$$k_i = \text{DPBG}(k, n)$$

Dove:

- **k** è la chiave privata della comunicazione.
- **n** è il **nonce**, il problema permane se nessuno dei due viene aggiornato.

Trasparent Data Encryption (TDE)

Nel caso in cui si voglia cifrare un disco, si vuole non inficiare lo spazio totale che si ha, quindi per evitare che il nonce venga salvato per ogni porzione scritta su disco normalmente si tende ad utilizzare informazioni che esistono già.
Il contesto non può essere utilizzato in quanto nel tempo non cambia mai quindi si è comunque affetti da *key reuse*.

2.4 Block Cipher & Modes of Operation

Un **cifrario a blocchi** è una famiglia di permutazioni pseudo-causali con chiave [***keyed family of pseudorandom permutation (PRP)***].

$$\mathbf{F} \ : \ \{0,1\}^{Lk} \times \{0,1\}^{Lb} \mapsto \{0,1\}^{Lb}$$

Dove **Lb** è la lunghezza del blocco da cifrare, mentre **Lk** è la lunghezza della chiave. Nei *block cipher* sia la funzione F che F^{-1} sono **efficenti**