

CRYPTO1 Stream Cypher

Stefano Fontana, Mat.727199

A.A. 2021/2022

Università degli Studi di Brescia

2022-03-24

CRYPTO1

CRYPTO1 Stream Cypher

Stefano Fontana, Mat.727199
A.A. 2021/2022
Università degli Studi di Brescia

Sommario

Sommario i

1 Sommario

2 ISO14443

3 Cifrari di flusso

4 CRYPTO1

5 Bibliografia

1. Sommario

2. ISO14443

UUID e anticollisione

Tag MIFARE Classic

Struttura della memoria

Lettura e scrittura

Three pass authentication sequence

3. Cifrari di flusso

Definizione

Struttura

2022-03-24

CRYPTO1

Sommario

Sommario

Sommario i

1. Sommario

2. ISO14443

UUID e anticollisione

Tag MIFARE Classic

Struttura della memoria

Lettura e scrittura

Three pass authentication sequence

3. Cifrari di flusso

Definizione

Struttura

Sommario ii

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

5 Bibliogra-
fia

4. CRYPTO1

Storia

Vulnerabilità

Reverse Engineering

Filter Function

Attacchi

Genuine Reader

Genuine Tag

Nested Authentication

2022-03-24

CRYPTO1

Sommario

Sommario

Sommario ii

4. CRYPTO1

Storia

Vulnerabilità

Reverse Engineering

Filter Function

Attacchi

Genuine Reader

Genuine Tag

Nested Authentication

Sommario iii

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

5 Bibliogra-
fia

Considerazioni

RNG

TRNG

LFSR

Grain128

Successori

5. Bibliografia

2022-03-24

CRYPTO1

Sommario

Sommario

Sommario iii

Considerazioni

RNG

TRNG

LFSR

Grain128

Successori

5. Bibliografia

ISO14443

2022-03-24
CRYPTO1
ISO14443

ISO14443

ISO14443a e NFC

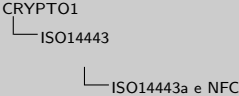
Cos’è NFC

Tecnologia di comunicazione contactless (short range wireless) che permette la produzione di dispositivi a basso costo[COO13]

Come funziona

- Semplificando il tutto, la comunicazione NFC è equivalente a un trasformatore.
- La trasmissione dell'informazione avviene mediante ASK e Load Modulation[Ins14]

2022-03-24



Cos'è NFC

Tecnologia di comunicazione contactless (short range wireless) che permette la produzione di dispositivi a basso costo[COO13]

Come funziona

- Semplificando il tutto, la comunicazione NFC è equivalente a un trasformatore.
- La trasmissione dell'informazione avviene mediante ASK e Load Modulation[Ins14]

ISO14443a e NFC

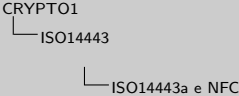
Cos’è NFC

Tecnologia di comunicazione contactless (short range wireless) che permette la produzione di dispositivi a basso costo[COO13]

Come funziona

- Semplificando il tutto, la comunicazione NFC è equivalente a un trasformatore.
- La trasmissione dell’informazione avviene mediante ASK e Load Modulation[Ins14]

2022-03-24



La codifica dell’informazione avviene con una variante di Manchester Encoding[Ins14]

Uno dei principali vantaggi della tecnologia è che il tag può essere completamente passivo e alimentarsi grazie al campo magnetico generato dal lettore

che causa induzione magnetica nella bobina del dispositivo.

Cos’è NFC
Tecnologia di comunicazione contactless (short range wireless) che permette la produzione di dispositivi a basso costo[COO13]

Come funziona

- Semplificando il tutto, la comunicazione NFC è equivalente a un trasformatore.
- La trasmissione dell’informazione avviene mediante ASK e Load Modulation[Ins14]

- 1 Sommario
- 2 ISO14443
 - UUID e anticollisione
 - Tag MIFARE Classic
- 3 Cifrari di flusso
- 4 CRYPTO1
- 5 Bibliografia

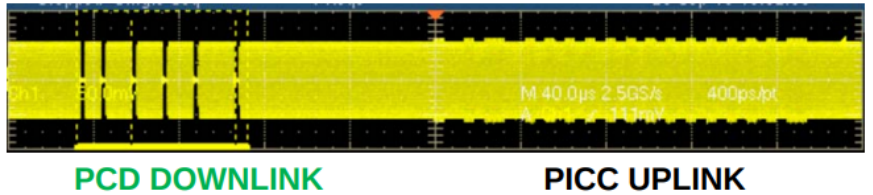


Figura 1: Traccia della comunicazione tra dispositivo e lettore [Ins14]

Si noti come la frequenza della portante sia notevolmente elevata

Ciò permette di avere dei dispositivi (TAG) completamente passivi alimentati dalla portante stessa.

CRYPTO1
ISO14443
2022-03-24



Figura 1: Traccia della comunicazione tra dispositivo e lettore [Ins14]

Si noti come la frequenza della portante sia notevolmente elevata

Ciò permette di avere dei dispositivi (TAG) completamente passivi alimentati dalla portante stessa.

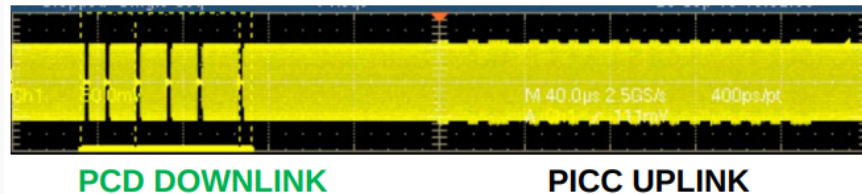


Figura 1: Traccia della comunicazione tra dispositivo e lettore [Ins14]

Si noti come la frequenza della portante sia notevolmente elevata

Ciò permette di avere dei dispositivi (TAG) completamente passivi alimentati dalla portante stessa.



Figura 1: Traccia della comunicazione tra dispositivo e lettore [Ins14]

Si noti come la frequenza della portante sia notevolmente elevata
Ciò permette di avere dei dispositivi (TAG) completamente passivi alimentati dalla portante stessa.

Dall'immagine è possibile osservare come la trasmissione in downlink sia 100% ASK, dove le tempistiche di bit sono notevolmente inferiori ai 13.56MHz della portante. Ciò permette al dispositivo di essere completamente passivo e di auto alimentarsi tramite l'effetto trasformatore causato dal coupling elettromagnetico delle due induttanze. La trasmissione inversa, dal tag al lettore, di conseguenza non può avvenire allo stesso modo, essendo il tag non in grado di generare una portante.

La soluzione è di variare l'impedenza collegata alla spira di ricezione nel tag in modo da utilizzare più corrente. Questo genererà una maggiore corrente nel lato trasmettitore che, mediante la resistenza interna di quest'ultimo, causerà una caduta di tensione leggibile e interpretabile.

UUID e identificazione multipla di TAG

1 Sommario

2 ISO14443

UUID e
anticollisione

Tag MIFARE
Classic

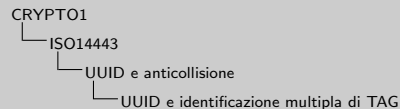
3 Cifrari di
flusso

4 CRYPTO1

5 Bibliogra-
fia

- L'UUID è un codice di univoco che permette l'identificazione dei tag. Un generico tag ISO14443-compliant possiede un UUID di 10byte, ma varie implementazioni permettono di avere una lunghezza a partire da 4byte.[NXP18a]
- Mediante il ciclo di identificazione e anticollisione è possibile ottenere la lista di tutti i tag presenti nelle vicinanze del lettore.
- Sarà poi possibile inviare comandi specifici a un solo tag mediante il processo di selezione

2022-03-24



UUID e identificazione multipla di TAG

- L'UUID è un codice di univoco che permette l'identificazione dei tag. Un generico tag ISO14443-compliant possiede un UUID di 10byte, ma varie implementazioni permettono di avere una lunghezza a partire da 4byte.[NXP18a]
- Mediante il ciclo di identificazione e anticollisione è possibile ottenere la lista di tutti i tag presenti nelle vicinanze del lettore.
- Sarà poi possibile inviare comandi specifici a un solo tag mediante il processo di selezione

UUID e identificazione multipla di TAG

1 Sommario

2 ISO14443

UUID e
anticollisione

Tag MIFARE
Classic

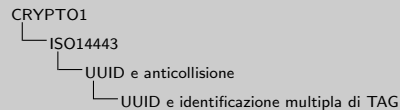
3 Cifrari di
flusso

4 CRYPTO1

5 Bibliogra-
fia

- L'UUID è un codice di univoco che permette l'identificazione dei tag. Un generico tag ISO14443-compliant possiede un UUID di 10byte, ma varie implementazioni permettono di avere una lunghezza a partire da 4byte.[NXP18a]
- Mediante il ciclo di identificazione e anticollisione è possibile ottenere la lista di tutti i tag presenti nelle vicinanze del lettore.
- Sarà poi possibile inviare comandi specifici a un solo tag mediante il processo di selezione

2022-03-24



UUID e identificazione multipla di TAG

- L'UUID è un codice di univoco che permette l'identificazione dei tag. Un generico tag ISO14443-compliant possiede un UUID di 10byte, ma varie implementazioni permettono di avere una lunghezza a partire da 4byte.[NXP18a]
- Mediante il ciclo di identificazione e anticollisione è possibile ottenere la lista di tutti i tag presenti nelle vicinanze del lettore.
- Sarà poi possibile inviare comandi specifici a un solo tag mediante il processo di selezione

UUID e identificazione multipla di TAG

1 Sommario

2 ISO14443

UUID e
anticollisione

Tag MIFARE
Classic

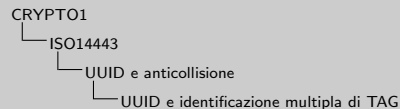
3 Cifrari di
flusso

4 CRYPTO1

5 Bibliogra-
fia

- L'UUID è un codice di univoco che permette l'identificazione dei tag. Un generico tag ISO14443-compliant possiede un UUID di 10byte, ma varie implementazioni permettono di avere una lunghezza a partire da 4byte.[NXP18a]
- Mediante il ciclo di identificazione e anticollisione è possibile ottenere la lista di tutti i tag presenti nelle vicinanze del lettore.
- Sarà poi possibile inviare comandi specifici a un solo tag mediante il processo di selezione

2022-03-24



UUID e identificazione multipla di TAG

- L'UUID è un codice di univoco che permette l'identificazione dei tag. Un generico tag ISO14443-compliant possiede un UUID di 10byte, ma varie implementazioni permettono di avere una lunghezza a partire da 4byte.[NXP18a]
- Mediante il ciclo di identificazione e anticollisione è possibile ottenere la lista di tutti i tag presenti nelle vicinanze del lettore.
- Sarà poi possibile inviare comandi specifici a un solo tag mediante il processo di selezione

Secondo lo standard, durante il processo di discover (Denominato “anticollision loop”) è possibile distinguere i tag grazie ai loro ID. È interessante notare, leggendo [NXP18b], che sono possibili più iterazioni del ciclo di anticollisione (CL1, CL2, CL3) dove ogni esecuzione incrementa il numero di byte che definiscono l'UUID. Risulta quindi che per una carta implementante tutti e tre i livelli di anticollisione l'UUID sarà **univoco** e lungo 80 bit; mentre per le carte con soli due livelli l'UUID sarà di 56 bit ma pur sempre **univoco**. L'unica implementazione in cui **non è garantito che il valore sia univoco** è quella base da 32 bit.

Il protocollo poi prosegue con un paradigma “select and operate”, dove una sessione di comunicazione viene iniziata tramite l'UUID inviato dal lettore.

A questo punto tutti i tag nella zona attiva rimarranno in stato IDLE ad eccezione del tag che è stato interpellato.

MIFARE Classic: Struttura della memoria

I tag MIFARE Classic sono tra i più **semplici** ed **economici**:

Il tag consiste in un piccolo frontend radio e logico che possa gestire la comunicazione e in una memoria non volatile dove salvare le configurazioni.[NXP18b]

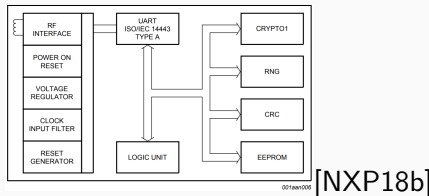
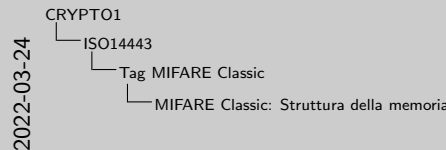


Figura 2: Diagramma dei componenti interni a un chip MIFARE Classic



La struttura di un tag MIFARE CLASSIC è da ritenersi interessante. Dato il ridotto costo del dispositivo ne consegue una bassa complessità elettronica. Infatti esso è costituito per una buona parte da componenti standard atti alla gestione della comunicazione e del chip in sè: L'interfaccia radio e il regolatore di tensione infatti sono gli unici componenti analogici presenti sul silicio. Successivamente si ha l'unità logica che ha il compito di gestirà la comunicazione e vigilare sulle operazioni, ma quest'ultima non ha grande complessità. L'algoritmo utilizzato è di fatto molto contenuto ed è equivalente a una macchina a stati. È interessante notare come la gestione della

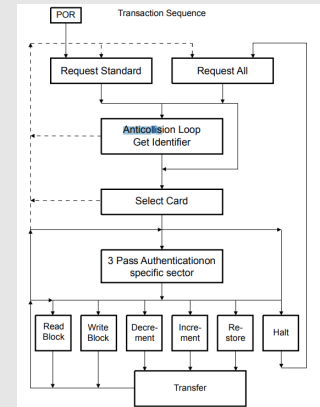
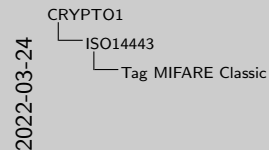


Figura 3: Schema a blocchi della gestione

La memoria è divisa in **settori** e **blocchi**:

- 16 settori [0–15]
- 4 blocchi per settore [0–3]
- Solo tre blocchi per settore possono includere dati [0–2]
- Il blocco 0:0 è leggibile senza autenticazione, protetto in scrittura e contiene l'UUID e dati del produttore
- I blocchi x:3 contengono le chiavi di scrittura e lettura (*KeyA* e *KeyB*), oltre che gli indicatori di protezione (access bits)



La memoria è divisa in **settori** e **blocchi**:

- 16 settori [0–15]
- 4 blocchi per settore [0–3]
- Solo tre blocchi per settore possono includere dati [0–2]
- Il blocco 0:0 è leggibile senza autenticazione, protetto in scrittura e contiene l'UUID e dati del produttore
- I blocchi x:3 contengono le chiavi di scrittura e lettura (*KeyA* e *KeyB*), che gli indicatori di protezione (access bits)

La memoria è divisa in **settori** e **blocchi**:

- 16 settori [0–15]
- 4 blocchi per settore [0–3]
- Solo tre blocchi per settore possono includere dati [0–2]
- Il blocco 0:0 è leggibile senza autenticazione, protetto in scrittura e contiene l'UUID e dati del produttore
- I blocchi x:3 contengono le chiavi di scrittura e lettura (*KeyA* e *KeyB*), oltre che gli indicatori di protezione (access bits)

La memoria è divisa in **settori** e **blocchi**:

- 16 settori [0–15]
- 4 blocchi per settore [0–3]
- Solo tre blocchi per settore possono includere dati [0–2]

• Il blocco 0:0 è leggibile senza autenticazione, protetto in scrittura e contiene l'UUID e dati del produttore

• I blocchi x:3 contengono le chiavi di scrittura e lettura (*KeyA* e *KeyB*), che gli indicatori di protezione (access bits)

La memoria è divisa in **settori** e **blocchi**:

- 16 settori [0–15]
- 4 blocchi per settore [0–3]
- Solo tre blocchi per settore possono includere dati [0–2]
- Il blocco 0:0 è leggibile senza autenticazione, protetto in scrittura e contiene l'UUID e dati del produttore
- I blocchi x:3 contengono le chiavi di scrittura e lettura (*KeyA* e *KeyB*), oltre che gli indicatori di protezione (access bits)

La memoria è divisa in **settori** e **blocchi**:

- 16 settori [0–15]
- 4 blocchi per settore [0–3]
- Solo tre blocchi per settore possono includere dati [0–2]
- Il blocco 0:0 è leggibile senza autenticazione, protetto in scrittura e contiene l'UUID e dati del produttore
- I blocchi x:3 contengono le chiavi di scrittura e lettura (*KeyA* e *KeyB*), che gli indicatori di protezione (access bits)

La memoria è divisa in **settori** e **blocchi**:

- 16 settori [0–15]
- 4 blocchi per settore [0–3]
- Solo tre blocchi per settore possono includere dati [0–2]
- Il blocco 0:0 è leggibile senza autenticazione, protetto in scrittura e contiene l'UUID e dati del produttore
- I blocchi x:3 contengono le chiavi di scrittura e lettura (*KeyA* e *KeyB*), oltre che gli indicatori di protezione (access bits)



È molto interessante notare come, al fine di mantenere bassi i costi di fabbricazione, le memorie sono le responsabili dei i costi (e lo spazio su silicio) più elevati.

Di conseguenza all'interno dei tag la memoria stessa viene utilizzata per il salvataggio delle chiavi di accesso e delle condizioni.

È inoltre interessante notare come esistano due diverse chiavi di cifratura per ogni settore: questo è fatto al fine di permettere l'utilizzo diversificato del tag a fronte di un segreto condiviso diverso. Molte volte la chiave A è utilizzata per permettere la sola lettura dei blocchi (escluso quello contenente la chiave B), mentre la chiave B è una chiave master in grado di riprogrammare il tag.

Bisogna però citare [Cou09], dove l'autore sottolinea come l'implementazione dell'algoritmo sia una “waste of silicon” data la caratteristica dell'offu-

scamento, dove funzioni identità sono implementate in modi diversi sul die. Questo, moltiplicato per il numero di tag prodotti causa un vero e proprio

spreco di materiale.

La memoria è divisa in **settori** e **blocchi**:

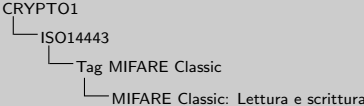
- 16 settori [0–15]
- 4 blocchi per settore [0–3]
- Solo tre blocchi per settore possono includere dati [0–2]
- Il blocco 0:0 è leggibile senza autenticazione, protetto in scrittura e contiene l'UUID e dati del produttore
- I blocchi x:3 contengono le chiavi di scrittura e lettura (*KeyA* e *KeyB*), oltre che gli indicatori di protezione (access bits)

MIFARE Classic: Lettura e scrittura

Prima di effettuare qualunque azione sulla memoria è necessario autenticarsi con la chiave adatta al settore in questione.

Il processo di autenticazione è chiamato “*Three pass authentication sequence*”[NXP18b] e sfrutta un cifrario a flusso

2022-03-24



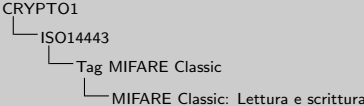
Prima di effettuare qualunque azione sulla memoria è necessario autenticarsi con la chiave adatta al settore in questione.
Il processo di autenticazione è chiamato “*Three pass authentication sequence*”[NXP18b] e sfrutta un cifrario a flusso

MIFARE Classic: Lettura e scrittura

Prima di effettuare qualunque azione sulla memoria è necessario autenticarsi con la chiave adatta al settore in questione.

Il processo di autenticazione è chiamato “*Three pass authentication sequence*”[NXP18b] e sfrutta un cifrario a flusso

2022-03-24



Durante questa procedura il cifrario viene inizializzato in uno stato comune al fine di permettere una trasmissione riservata. Lo stato condiviso sfrutta

la presenza di nonce casuali per assicurare l'unicità della comunicazione in modo da impedire correlazioni e attacchi replica.

Prima di effettuare qualunque azione sulla memoria è necessario autenticarsi con la chiave adatta al settore in questione.
Il processo di autenticazione è chiamato “*Three pass authentication sequence*”[NXP18b] e sfrutta un cifrario a flusso

Three pass authentication sequence[GKGM⁺08]

1 Sommario

2 ISO14443

UUID e
anticollisione

Tag MIFARE
Classic

Struttura della
memoria

Lettura e
scrittura

Three pass
authentication
sequence

3 Cifrari di
flusso

4 CRYPTO1

5 Bibliogra-
fia

- Il tag entra nel campo magnetico del lettore e si accende

- Protocollo di anticollisione (*Non descritto*) e invio dell'UUID

- Il lettore effettua una richiesta di autenticazione al blocco richiesto

- Il tag ritorna un nonce n_t e lo trasmette in chiaro

- Il lettore quindi invia il proprio nonce n_r e la risposta alla challenge a_r cifrandoli mediante xoring con lo stream proveniente dal cifrario a flusso ks_1 e ks_2

- L'autenticazione si conclude con il tag che risponderà alla challenge del reader con a_t cifrato tramite ks_3

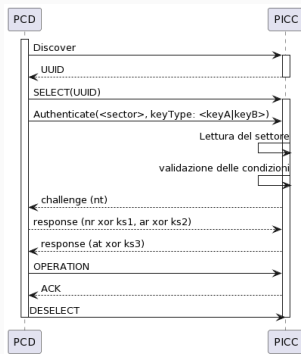
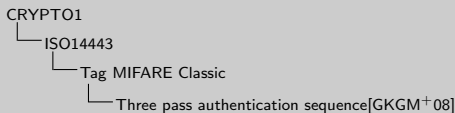


Figura 4: Diagramma di sequenza del processo di autenticazione a un blocco

2022-03-24



Three pass authentication sequence[GKGM⁺08]

- Il tag entra nel campo magnetico del lettore e si accende
- Protocollo di anticollisione (*Non descritto*) e invio dell'UUID
- Il lettore effettua una richiesta di autenticazione al blocco richiesto
- Il tag ritorna un nonce n_t e lo trasmette in chiaro
- Il lettore quindi invia il proprio nonce n_r e la risposta alla challenge a_r cifrandoli mediante xoring con lo stream proveniente dal cifrario a flusso ks_1 e ks_2
- L'autenticazione si conclude con il tag che risponderà alla challenge del reader con a_t cifrato tramite ks_3



Figura 4: Diagramma di sequenza del processo di autenticazione a un blocco

Three pass authentication sequence[GKGM⁺08]

1 Sommario

2 ISO14443

UUID e
anticollisione

Tag MIFARE
Classic

Struttura della
memoria

Lettura e
scrittura

Three pass
authentication
sequence

3 Cifrari di
flusso

4 CRYPTO1

5 Bibliogra-
fia

- Il tag entra nel campo magnetico del lettore e si accende
- Protocollo di anticollisione (*Non descritto*) e invio dell'UUID
- Il lettore effettua una richiesta di autenticazione al blocco richiesto
- Il tag ritorna un nonce n_t e lo trasmette in chiaro
- Il lettore quindi invia il proprio nonce n_r e la risposta alla challenge a_r cifrandoli mediante xoring con lo stream proveniente dal cifrario a flusso ks_1 e ks_2
- L'autenticazione si conclude con il tag che risponderà alla challenge del reader con a_t cifrato tramite ks_3

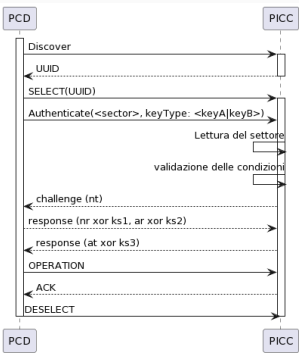
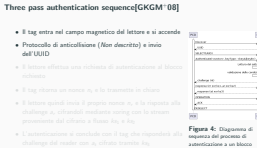
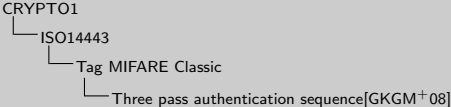


Figura 4: Diagramma di sequenza del processo di autenticazione a un blocco

2022-03-24



Three pass authentication sequence[GKGM⁺08]

- Il tag entra nel campo magnetico del lettore e si accende
- Protocollo di anticollisione (*Non descritto*) e invio dell'UUID
- Il lettore effettua una richiesta di autenticazione al blocco richiesto
- Il tag ritorna un nonce n_t e lo trasmette in chiaro
- Il lettore quindi invia il proprio nonce n_r e la risposta alla challenge a_r cifrandoli mediante xoring con lo stream proveniente dal cifrario a flusso ks_1 e ks_2
- L'autenticazione si conclude con il tag che risponderà alla challenge del reader con a_t cifrato tramite ks_3

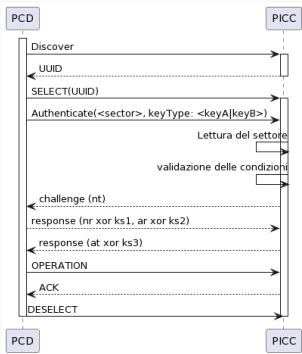
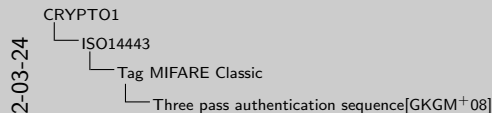


Figura 4: Diagramma di sequenza del processo di autenticazione a un blocco



Three pass authentication sequence[GKGM⁺08]

- Il tag entra nel campo magnetico del lettore e si accende
- Protocollo di anticollisione (*Non descritto*) e invio dell'UUID
- Il lettore effettua una richiesta di autenticazione al blocco richiesto
- Il tag ritorna un nonce n_t e lo trasmette in chiaro
- Il lettore quindi invia il proprio nonce n_r e la risposta alla challenge a_r cifrandoli mediante xoring con lo stream proveniente dal cifrario a flusso ks_1 e ks_2
- L'autenticazione si conclude con il tag che risponderà alla challenge del reader con a_t cifrato tramite ks_3



Figura 4: Diagramma di sequenza del processo di autenticazione a un blocco

Three pass authentication sequence[GKGM⁺08]

1 Sommario

2 ISO14443

UUID e
anticollisione

Tag MIFARE
Classic

Struttura della
memoria

Lettura e
scrittura

Three pass
authentication
sequence

3 Cifrari di
flusso

4 CRYPTO1

5 Bibliogra-
fia

- Il tag entra nel campo magnetico del lettore e si accende
- Protocollo di anticollisione (*Non descritto*) e invio dell'UUID
- Il lettore effettua una richiesta di autenticazione al blocco richiesto
- Il tag ritorna un nonce n_t e lo trasmette in chiaro
- Il lettore quindi invia il proprio nonce n_r e la risposta alla challenge a_r cifrandoli mediante xoring con lo stream proveniente dal cifrario a flusso ks_1 e ks_2
- L'autenticazione si conclude con il tag che risponderà alla challenge del reader con a_t cifrato tramite ks_3

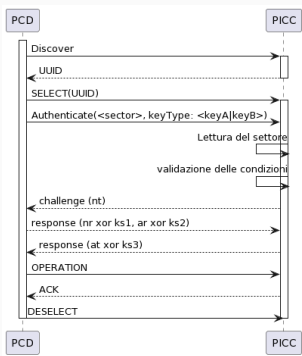
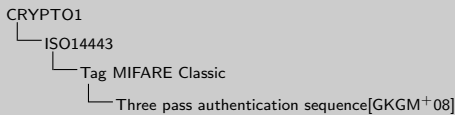


Figura 4: Diagramma di sequenza del processo di autenticazione a un blocco

2022-03-24



Three pass authentication sequence[GKGM⁺08]

- Il tag entra nel campo magnetico del lettore e si accende
- Protocollo di anticollisione (*Non descritto*) e invio dell'UUID
- Il lettore effettua una richiesta di autenticazione al blocco richiesto
- Il tag ritorna un nonce n_t e lo trasmette in chiaro
- Il lettore quindi invia il proprio nonce n_r e la risposta alla challenge a_r cifrandoli mediante xoring con lo stream proveniente dal cifrario a flusso ks_1 e ks_2
- L'autenticazione si conclude con il tag che risponderà alla challenge del reader con a_t cifrato tramite ks_3



Figura 4: Diagramma di sequenza del processo di autenticazione a un blocco

Three pass authentication sequence[GKGM⁺08]

- Il tag entra nel campo magnetico del lettore e si accende
- Protocollo di anticollisione (*Non descritto*) e invio dell'UUID
- Il lettore effettua una richiesta di autenticazione al blocco richiesto
- Il tag ritorna un nonce n_t e lo trasmette in chiaro
- Il lettore quindi invia il proprio nonce n_r e la risposta alla challenge a_r cifrandoli mediante xoring con lo stream proveniente dal cifrario a flusso ks_1 e ks_2
- L'autenticazione si conclude con il tag che risponderà alla challenge del reader con a_t cifrato tramite ks_3

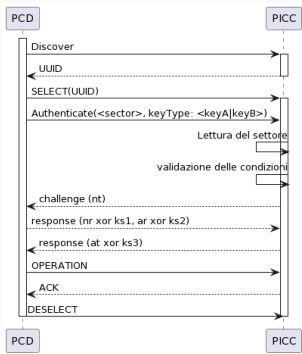
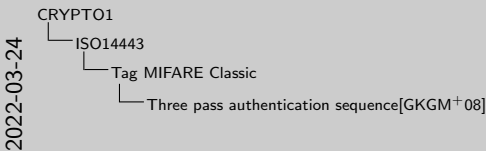


Figura 4: Diagramma di sequenza del processo di autenticazione a un blocco



Three pass authentication sequence[GKGM⁺08]

- Il tag entra nel campo magnetico del lettore e si accende
- Protocollo di anticollisione (*Non descritto*) e invio dell'UUID
- Il lettore effettua una richiesta di autenticazione al blocco richiesto
- Il tag ritorna un nonce n_t e lo trasmette in chiaro
- Il lettore quindi invia il proprio nonce n_r e la risposta alla challenge a_r cifrandoli mediante xoring con lo stream proveniente dal cifrario a flusso ks_1 e ks_2

* L'autenticazione si conclude con il tag che risponderà alla challenge del reader con a_t cifrato tramite ks_3

Figura 4: Diagramma di sequenza del processo di autenticazione a un blocco

Three pass authentication sequence[GKGM⁺08]

1 Sommario

2 ISO14443

UUID e
anticollisione

Tag MIFARE
Classic

Struttura della
memoria

Lettura e
scrittura

Three pass
authentication
sequence

3 Cifrari di
flusso

4 CRYPTO1

5 Bibliogra-
fia

- Il tag entra nel campo magnetico del lettore e si accende
- Protocollo di anticollisione (*Non descritto*) e invio dell'UUID
- Il lettore effettua una richiesta di autenticazione al blocco richiesto
- Il tag ritorna un nonce n_t e lo trasmette in chiaro
- Il lettore quindi invia il proprio nonce n_r e la risposta alla challenge a_r cifrandoli mediante xoring con lo stream proveniente dal cifrario a flusso ks_1 e ks_2
- L'autenticazione si conclude con il tag che risponderà alla challenge del reader con a_t cifrato tramite ks_3

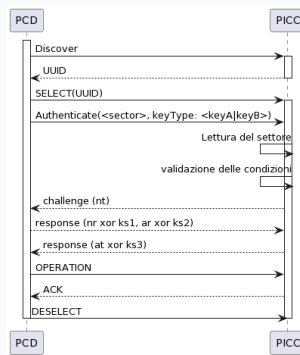
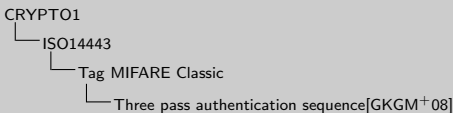


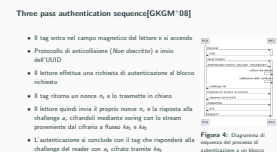
Figura 4: Diagramma di sequenza del processo di autenticazione a un blocco

2022-03-24



Come è possibile vedere dallo schema, il processo di autenticazione inizia con il tag che invia un proprio nonce n_t al lettore, in modo da - idealmente - impedire il riutilizzo di uno stato interno. Infatti il nonce è utilizzato in concomitanza con l'UUID e la chiave del settore per inizializzare il cifrario. Infatti senza UUID e senza nonce, due tag con chiave uguale potrebbero essere scoperti solamente ascoltando (eavesdropping) le comunicazioni. Inserendo l'UUID nell'algoritmo ciò diviene impossibile perchè i tag avranno uno stato diverso in funzione dell'UUID, il quale è unico (o comunque è improbabile che due tag abbiano lo stesso uuid) Ciò non ferma però eventuali attacchi che fanno leva sulla ripetizione degli stati iniziali del tag.

A tal fine viene introdotto il numero casuale deciso dal tag: la sua trasmissione in chiaro non è direttamente un vettore di attacco, ma si vedrà che è stato il metodo utilizzato per scoprire svariate vulnerabilità sull'rng.



Three pass authentication sequence[GKGM⁺08] i

La scelta dei nonce avviene come dalla seguente illustrazione, dove *key* è la chiave condivisa (ovvero la chiave che il lettore usa per autenticarsi)

Generazione del tag nonce

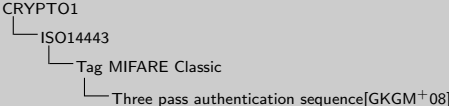
$$n_t = nextRandom()$$

$$send(n_t)$$

Dopo l’invio, sia il tag che il lettore inizializzano il proprio cifrario a flusso e computano i primi 32bit del *keystream*

$$ks_1 = cipherInit(key, uid, n_t)$$

2022-03-24



Three pass authentication sequence[GKGM⁺08] i

La scelta dei nonce avviene come dalla seguente illustrazione, dove *key* è la chiave condivisa (ovvero la chiave che il lettore usa per autenticarsi)

Generazione del tag nonce

```
n_t = nextRandom()
send(n_t)
```

Dopo l’invio, sia il tag che il lettore inizializzano il proprio cifrario a flusso e computano i primi 32bit del *keystream*

```
ks_1 = cipherInit(key, uid, n_t)
```

1	Sommario
2	ISO14443
	UUID e anticollisione
	Tag MIFARE Classic
	Struttura della memoria
	Lettura e scrittura
	Three pass authentication sequence

3	Cifrari di flusso
---	-------------------

4	CRYPTO1
---	---------

5	Bibliografia
---	--------------

Three pass authentication sequence[GKGM⁺08] ii

Generazione del reader nonce

$$n_r = nextRandom()$$

Invio del reader nonce (cifrato) e della risposta alla challenge così computata

$$a_t = suc^2(n_t)$$

$$ks_2 = cipher(n_r)$$

$$send(n_r \oplus ks_1); send(a_t \oplus ks_2)$$

Risposta finale

$$send(suc^3(n_t) \oplus ks_3)$$



- Sia la funzione $suc(v)$ la successiva iterazione del rng con seed v
Il processo di scambio di chiavi viene così descritto: [NXP18b]
- il tag sceglie il proprio nonce n_t e lo invia al lettore
 - entrambe le parti inizializzano il proprio cifrario con una funzione della chiave key , n_t e l'UUID del tag
 - una volta ottenuto il nonce, il lettore computa i primi 32 bit del keystream, inserendo nel LFSR i 32 bit di una funzione $f(n_t, uuid)$
 - il lettore genera n_r e lo invia al tag cifrandolo con ks_1 ($n_r \oplus ks_1$); insieme invia la risposta alla challenge $a_r = suc^2(n_t)$, seconda iterazione del RNG con seed n_t , cifrata con i secondi 32 bit ottenuti aggiornando il proprio cifrario con n_r
 - il tag, quindi, aggiorna il proprio cifrario con n_r e verifica che a_r sia valida.
 - per completare l'autenticazione viene quindi inviato $suc^3(n_t)$ cifrato con k_3 in modo che Il lettore possa validare il tag.

Cifrari di flusso

2022-03-24

CRYPTO1
Cifrari di flusso

Cifrari di flusso

Cifrari di flusso

1 Sommario

2 ISO14443

3 Cifrari di
flusso

Definizione

Struttura

4 CRYPTO1

5 Bibliogra-
fia

Un cifrario a flusso è un dispositivo in grado di generare una sequenza di bit dipendente dallo stato iniziale dello stesso con le seguenti proprietà:

- La periodicità della sequenza di bit deve essere elevata (idealmente infinita)
- Non deve essere possibile ricavare lo stato interno del cifrario data una porzione dello stream di output.
- Il bitstream non deve essere riconoscibile rispetto a un rumore randomico, ovvero la probabilità che ciascun bit possa essere 0 o 1 non deve dipendere dai bit precedenti e deve essere uguale a 0.5
- non devono esserci “*Weak Keys*” o “*Weak States*”



Cifrari di flusso

Un cifrario a flusso è un dispositivo in grado di generare una sequenza di bit dipendente dallo stato iniziale dello stesso con le seguenti proprietà:

- La periodicità della sequenza di bit deve essere elevata (idealmente infinita)
- Non deve essere possibile ricavare lo stato interno del cifrario data una porzione dello stream di output.
- Il bitstream non deve essere riconoscibile rispetto a un rumore randomico, ovvero la probabilità che ciascun bit possa essere 0 o 1 non deve dipendere dai bit precedenti e deve essere uguale a 0.5
- non devono esserci “*Weak Keys*” o “*Weak States*”

Cifrari di flusso

1 Sommario

2 ISO14443

3 Cifrari di
flusso

Definizione

Struttura

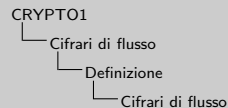
4 CRYPTO1

5 Bibliogra-
fia

Un cifrario a flusso è un dispositivo in grado di generare una sequenza di bit dipendente dallo stato iniziale dello stesso con le seguenti proprietà:

- La periodicità della sequenza di bit deve essere elevata (idealmente infinita)
- Non deve essere possibile ricavare lo stato interno del cifrario data una porzione dello stream di output.
- Il bitstream non deve essere riconoscibile rispetto a un rumore randomico, ovvero la probabilità che ciascun bit possa essere 0 o 1 non deve dipendere dai bit precedenti e deve essere uguale a 0.5
- non devono esserci “*Weak Keys*” o “*Weak States*”

2022-03-24



Cifrari di flusso

Un cifrario a flusso è un dispositivo in grado di generare una sequenza di bit dipendente dallo stato iniziale dello stesso con le seguenti proprietà:

- La periodicità della sequenza di bit deve essere elevata (idealmente infinita)
- Non deve essere possibile ricavare lo stato interno del cifrario data una porzione dello stream di output.
- Il bitstream non deve essere riconoscibile rispetto a un rumore randomico, ovvero la probabilità che ciascun bit possa essere 0 o 1 non deve dipendere dai bit precedenti e deve essere uguale a 0.5
- non devono esserci “*Weak Keys*” o “*Weak States*”

Cifrari di flusso

1 Sommario

2 ISO14443

3 Cifrari di
flusso

Definizione

Struttura

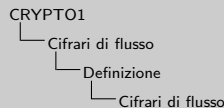
4 CRYPTO1

5 Bibliogra-
fia

Un cifrario a flusso è un dispositivo in grado di generare una sequenza di bit dipendente dallo stato iniziale dello stesso con le seguenti proprietà:

- La periodicità della sequenza di bit deve essere elevata (idealmente infinita)
- Non deve essere possibile ricavare lo stato interno del cifrario data una porzione dello stream di output.
- Il bitstream non deve essere riconoscibile rispetto a un rumore randomico, ovvero la probabilità che ciascun bit possa essere 0 o 1 non deve dipendere dai bit precedenti e deve essere uguale a 0.5
- non devono esserci “Weak Keys” o “Weak States”

2022-03-24



Cifrari di flusso

Un cifrario a flusso è un dispositivo in grado di generare una sequenza di bit dipendente dallo stato iniziale dello stesso con le seguenti proprietà:

- La periodicità della sequenza di bit deve essere elevata (idealmente infinita)
- Non deve essere possibile ricavare lo stato interno del cifrario data una porzione dello stream di output.
- Il bitstream non deve essere riconoscibile rispetto a un rumore randomico, ovvero la probabilità che ciascun bit possa essere 0 o 1 non deve dipendere dai bit precedenti e deve essere uguale a 0.5

• non devono esserci “Weak Keys” o “Weak States”

Cifrari di flusso

1 Sommario

2 ISO14443

3 Cifrari di
flusso

Definizione

Struttura

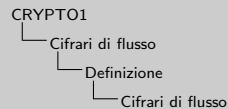
4 CRYPTO1

5 Bibliogra-
fia

Un cifrario a flusso è un dispositivo in grado di generare una sequenza di bit dipendente dallo stato iniziale dello stesso con le seguenti proprietà:

- La periodicità della sequenza di bit deve essere elevata (idealmente infinita)
- Non deve essere possibile ricavare lo stato interno del cifrario data una porzione dello stream di output.
- Il bitstream non deve essere riconoscibile rispetto a un rumore randomico, ovvero la probabilità che ciascun bit possa essere 0 o 1 non deve dipendere dai bit precedenti e deve essere uguale a 0.5
- non devono esserci “*Weak Keys*” o “*Weak States*”

2022-03-24



I cifrari di flusso prendono ispirazione da OneTimePad, dove il testo cifrato non è altro che lo xor bit a bit con una chiave. In sè OTP è un cifrario perfetto, dato che non permette a un ascoltatore di inferire sulle probabilità che i bit del testo sorgente siano 1 o 0; ciò è valido solo se la chiave è completamente randomica e non viene mai riutilizzata. Gli stream cipher sono stati dunque ideati al fine di generare continuamente bit con i quali cifrare la chiave, facendo sì che il bit non dipenda dai precedenti e che esso abbia uguale probabilità di essere in uno stato o nell'altro.

La convenienza degli stream ciphers sta nel fatto che ritornano lo stesso stream di dati a parità di stato iniziale.

Cifrari di flusso

Un cifrario a flusso è un dispositivo in grado di generare una sequenza di bit dipendente dallo stato iniziale dello stesso con le seguenti proprietà:

- La periodicità della sequenza di bit deve essere elevata (idealmente infinita)
- Non deve essere possibile ricavare lo stato interno del cifrario data una porzione dello stream di output.
- Il bitstream non deve essere riconoscibile rispetto a un rumore randomico, ovvero la probabilità che ciascun bit possa essere 0 o 1 non deve dipendere dai bit precedenti e deve essere uguale a 0.5
- non devono esserci “*Weak Keys*” o “*Weak States*”

Struttura

- 1 Sommario
- 2 ISO14443
- 3 Cifrari di flusso
 - Definizione
 - Struttura
- 4 CRYPTO1
- 5 Bibliografia

Il cifrario di flusso più semplice è un Linear Feedback Shift Register costituito da un solo blocco composto da:

- Un Registro a scorrimento dove, ad ogni iterazione, un bit può essere aggiunto “in coda” e il bit più significativo viene scartato.
- Una funzione $g(SR)$ che dato lo stato attuale computa il prossimo bit da aggiungere
- Una funzione di filtraggio $f(SR)$ che dallo stato del registro computa l'output.

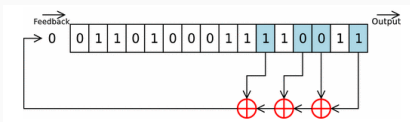


Figura 5: Diagramma di un LFSR dove $f(SR)$ è la funzione identità del bit finale

- CRYPTO1
 - Cifrari di flusso
 - Struttura
 - Struttura
- 2022-03-24

In questo caso la funzione g è

$$g(b_0, b_1, \dots, b_{16}) = b_{16} \oplus b_{14} \oplus b_{13} \oplus b_{11}$$

mentre

$$f(b_0, b_1, \dots, b_{16}) = b_{16}$$

Struttura

Il cifrario di flusso più semplice è un Linear Feedback Shift Register costituito da un solo blocco composto da:

- Un Registro a scorrimento dove, ad ogni iterazione, un bit può essere aggiunto “in coda” e il bit più significativo viene scartato.
- Una funzione $g(SR)$ che dato lo stato attuale computa il prossimo bit da aggiungere
- Una funzione di filtraggio $f(SR)$ che dallo stato del registro computa l'output.

Figura 5: Diagramma di un LFSR dove $f(SR)$ è la funzione identità del bit finale

Struttura

È possibile unire più LFSR in una funzione $F(LFSR1, \dots, LFSRN)$ di filtraggio generale come implementato da A5/1

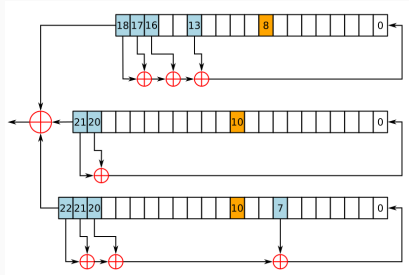


Figura 6: Diagramma di A5/1

2022-03-24

CRYPTO1
Cifrari di flusso
Struttura
Struttura

Struttura

È possibile unire più LFSR in una funzione $F(LFSR1, \dots, LFSRN)$ di filtraggio generale come implementato da A5/1



Figura 6: Diagramma di A5/1

Per il primo LFSR:

$$g_1(b_0 \dots b_{18}) = b_{18} \oplus b_{17} \oplus b_{16} \oplus b_{13}$$

$$f_1(b_0 \dots b_{18}) = b_{18}$$

Per il secondo LFSR:

$$g_2(b_0 \dots b_{21}) = b_{21} \oplus b_{20}$$

$$f_2(b_0 \dots b_{21}) = b_{21}$$

Per il terzo LFSR:

$$g_3(b_0 \dots b_{22}) = b_{22} \oplus b_{21} \oplus b_{20} \oplus b_7$$

$$f_3(b_0 \dots b_{22}) = b_{22}$$

Infine

$$F(r_1, r_2, r_3) = r_1 \oplus r_2 \oplus r_3$$

CRYPTO1

2022-03-24

CRYPTO1
CRYPTO1

CRYPTO1

Storia

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse
Engineering

Attacchi

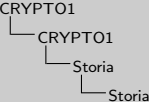
Considerazioni

Successori

5 Bibliogra-
fia

CRYPTO1 è uno stream cipher sviluppato da *NXP Semiconductors* nel 1994
insieme alla famiglia di tag MIFARE Classic [Tez17]

2022-03-24



Storia

CRYPTO1 è uno stream cipher sviluppato da *NXP Semiconductors* nel 1994
insieme alla famiglia di tag MIFARE Classic [Tez17]

Security through obscurity

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse
Engineering

Attacchi

Considerazioni

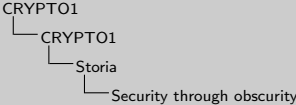
Successori

5 Bibliogra-
fia

La sicurezza del sistema era affidata al concetto di *Security through obscurity*

- Metodo fortemente sconsigliato da tutti gli organi normativi sulla sicurezza[SJT⁺08]
- Tecnica in contrasto con *Security by Design* e *Open security*

2022-03-24



Security through obscurity

La sicurezza del sistema era affidata al concetto di *Security through obscurity*

- Metodo fortemente sconsigliato da tutti gli organi normativi sulla sicurezza[SJT⁺08]
- Tecnica in contrasto con *Security by Design* e *Open security*

Security through obscurity

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse
Engineering

Attacchi

Considerazioni

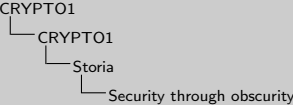
Successori

5 Bibliogra-
fia

La sicurezza del sistema era affidata al concetto di *Security through obscurity*

- Metodo fortemente sconsigliato da tutti gli organi normativi sulla sicurezza[SJT⁺08]
- Tecnica in contrasto con *Security by Design* e *Open security*

2022-03-24



Security through obscurity

La sicurezza del sistema era affidata al concetto di *Security through obscurity*

- Metodo fortemente sconsigliato da tutti gli organi normativi sulla sicurezza[SJT⁺08]

• Tecnica in contrasto con *Security by Design* e *Open security*

Security through obscurity

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse
Engineering

Attacchi

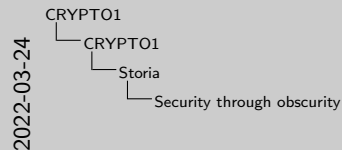
Considerazioni

Successori

5 Bibliogra-
fia

La sicurezza del sistema era affidata al concetto di *Security through obscurity*

- Metodo fortemente sconsigliato da tutti gli organi normativi sulla sicurezza[SJT⁺08]
- Tecnica in contrasto con *Security by Design* e *Open security*



Contrariamente alle due politiche *Security by Design* e *Open security* la sicurezza tramite offuscamento è fortemente sconsigliata, in quanto affida la sicurezza del sistema al fatto che nessuno riesca a comprenderlo.

Questa pratica rende quindi il sistema vulnerabile a qualsiasi attacco di tipo reverse engineering, oltre che a possibili fughe di informazioni.

L'utilizzo di ideologie "open" permette la validazione del sistema da parte di un maggior numero di enti e di membri di una comunità, favorendo così l'individuazione di falle in minor tempo.

Il metodo più efficiente, però, consiste sempre nell'utilizzo di sistemi già esistenti e ritenuti sicuri (p.e. tritium)

Security through obscurity

La sicurezza del sistema era affidata al concetto di *Security through obscurity*

- Metodo fortemente sconsigliato da tutti gli organi normativi sulla sicurezza[SJT⁺08]
- Tecnica in contrasto con *Security by Design* e *Open security*

La caduta di CRYPTO1

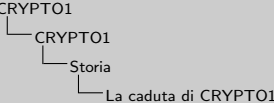
Nel 2008/2009 più ricercatori in contemporanea hanno trovato vulnerabilità e sono stati rilasciati attacchi sul critto-sistema CRYPTO1 che ne hanno interamente distrutto la sicurezza.[GKGM⁺08][CNO08][NESP08]

Il sistema presenta falle nella sicurezza in più settori:

- Random Number Generator
- Proprietà algebriche e vulnerabilità strutturali del LFSR
- Complessità delle chiavi (Che rendono il critto-sistema vulnerabile ad attacchi di tipo bruteforce [CNO08])
- Logica di gestione della comunicazione

(Praticamente presenta falle in ogni sua componente)

2022-03-24



La caduta di CRYPTO1

Nel 2008/2009 più ricercatori in contemporanea hanno trovato vulnerabilità e sono stati rilasciati attacchi sul critto-sistema CRYPTO1 che ne hanno interamente distrutto la sicurezza.[GKGM⁺08][CNO08][NESP08]

Il sistema presenta falle nella sicurezza in più settori:

- Random Number Generator
- Proprietà algebriche e vulnerabilità strutturali del LFSR
- Complessità delle chiavi (Che rendono il critto-sistema vulnerabile ad attacchi di tipo bruteforce [CNO08])
- Logica di gestione della comunicazione

(Praticamente presenta falle in ogni sua componente)

1 Sommario

2 ISO14443

3 Cifrari di flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse Engineering

Attacchi

Considerazioni

Successori

5 Bibliografia

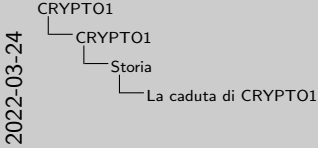
La caduta di CRYPTO1

Nel 2008/2009 più ricercatori in contemporanea hanno trovato vulnerabilità e sono stati rilasciati attacchi sul critto-sistema CRYPTO1 che ne hanno interamente distrutto la sicurezza.[GKGM⁺08][CNO08][NESP08]

Il sistema presenta falle nella sicurezza in più settori:

- Random Number Generator
- Proprietà algebriche e vulnerabilità strutturali del LFSR
- Complessità delle chiavi (Che rendono il critto-sistema vulnerabile ad attacchi di tipo bruteforce [CNO08])
- Logica di gestione della comunicazione

(Praticamente presenta falle in ogni sua componente)



La caduta di CRYPTO1

Nel 2008/2009 più ricercatori in contemporanea hanno trovato vulnerabilità e sono stati rilasciati attacchi sul critto-sistema CRYPTO1 che ne hanno interamente distrutto la sicurezza.[GKGM⁺08][CNO08][NESP08]

Il sistema presenta falle nella sicurezza in più settori:

- Random Number Generator
- Proprietà algebriche e vulnerabilità strutturali del LFSR
- Complessità delle chiavi (Che rendono il critto-sistema vulnerabile ad attacchi di tipo bruteforce [CNO08])
- Logica di gestione della comunicazione

(Praticamente presenta falle in ogni sua componente)

1 Sommario

2 ISO14443

3 Cifrari di flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse Engineering

Attacchi

Considerazioni

Successori

5 Bibliografia

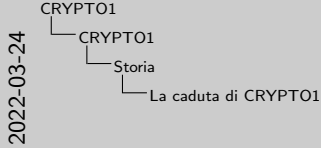
La caduta di CRYPTO1

Nel 2008/2009 più ricercatori in contemporanea hanno trovato vulnerabilità e sono stati rilasciati attacchi sul critto-sistema CRYPTO1 che ne hanno interamente distrutto la sicurezza.[GKGM⁺08][CNO08][NESP08]

Il sistema presenta falle nella sicurezza in più settori:

- Random Number Generator
- Proprietà algebriche e vulnerabilità strutturali del LFSR
- Complessità delle chiavi (Che rendono il critto-sistema vulnerabile ad attacchi di tipo bruteforce [CNO08])
- Logica di gestione della comunicazione

(Praticamente presenta falle in ogni sua componente)



La caduta di CRYPTO1

Nel 2008/2009 più ricercatori in contemporanea hanno trovato vulnerabilità e sono stati rilasciati attacchi sul critto-sistema CRYPTO1 che ne hanno interamente distrutto la sicurezza.[GKGM⁺08][CNO08][NESP08]

Il sistema presenta falle nella sicurezza in più settori:

- Random Number Generator
- Proprietà algebriche e vulnerabilità strutturali del LFSR
- Complessità delle chiavi (Che rendono il critto-sistema vulnerabile ad attacchi di tipo bruteforce [CNO08])
- Logica di gestione della comunicazione

(Praticamente presenta falle in ogni sua componente)

1 Sommario

2 ISO14443

3 Cifrari di flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse Engineering

Attacchi

Considerazioni

Successori

5 Bibliografia

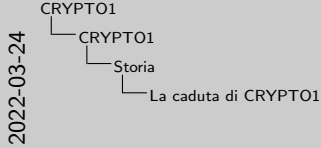
La caduta di CRYPTO1

Nel 2008/2009 più ricercatori in contemporanea hanno trovato vulnerabilità e sono stati rilasciati attacchi sul critto-sistema CRYPTO1 che ne hanno interamente distrutto la sicurezza.[GKGM⁺08][CNO08][NESP08]

Il sistema presenta falle nella sicurezza in più settori:

- Random Number Generator
- Proprietà algebriche e vulnerabilità strutturali del LFSR
- Complessità delle chiavi (Che rendono il critto-sistema vulnerabile ad attacchi di tipo bruteforce [CNO08])
- Logica di gestione della comunicazione

(Praticamente presenta falle in ogni sua componente)



La caduta di CRYPTO1

Nel 2008/2009 più ricercatori in contemporanea hanno trovato vulnerabilità e sono stati rilasciati attacchi sul critto-sistema CRYPTO1 che ne hanno interamente distrutto la sicurezza.[GKGM⁺08][CNO08][NESP08]

Il sistema presenta falle nella sicurezza in più settori:

- Random Number Generator
- Proprietà algebriche e vulnerabilità strutturali del LFSR
- Complessità delle chiavi (Che rendono il critto-sistema vulnerabile ad attacchi di tipo bruteforce [CNO08])
- Logica di gestione della comunicazione

(Praticamente presenta falle in ogni sua componente)

La caduta di CRYPTO1

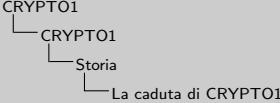
Nel 2008/2009 più ricercatori in contemporanea hanno trovato vulnerabilità e sono stati rilasciati attacchi sul critto-sistema CRYPTO1 che ne hanno interamente distrutto la sicurezza.[GKGM⁺08][CNO08][NESP08]

Il sistema presenta falle nella sicurezza in più settori:

- Random Number Generator
- Proprietà algebriche e vulnerabilità strutturali del LFSR
- Complessità delle chiavi (Che rendono il critto-sistema vulnerabile ad attacchi di tipo bruteforce [CNO08])
- Logica di gestione della comunicazione

(Praticamente presenta falle in ogni sua componente)

2022-03-24



La caduta di CRYPTO1

Nel 2008/2009 più ricercatori in contemporanea hanno trovato vulnerabilità e sono stati rilasciati attacchi sul critto-sistema CRYPTO1 che ne hanno interamente distrutto la sicurezza.[GKGM⁺08][CNO08][NESP08]

Il sistema presenta falle nella sicurezza in più settori:

- Random Number Generator
- Proprietà algebriche e vulnerabilità strutturali del LFSR
- Complessità delle chiavi (Che rendono il critto-sistema vulnerabile ad attacchi di tipo bruteforce [CNO08])
- Logica di gestione della comunicazione

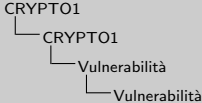
(Praticamente presenta falle in ogni sua componente)

Le vulnerabilità trovate negli anni nel critto-sistema sono le seguenti.

Alcune di queste sono di facile soluzione perché riguardano il lato hardware del lettore, per cui, a fronte di un costo maggiore, è possibile migliorarne le capacità.

Vulnerabilità

2022-03-24



1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse
Engineering

Attacchi

Considerazioni

Successori

5 Bibliogra-
fia

1. Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]

2. RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM⁺08] con condizione iniziale costante.

- 2.1 Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM⁺08][CNO08]
- 2.2 Inoltre i numeri casuali sono generati a partire da 16 bit del registro
- 2.3 In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell' attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)

3. L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM⁺08]

4. La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari

5. LFSR State Recovery

6. LFSR Rollback

7. I bit di parità sono computati sul plaintext e poi inviati non cifrati

Vulnerabilità

1. Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]
2. RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM⁺08] con condizione iniziale costante.
 - 2.1 Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM⁺08][CNO08]
 - 2.2 Inoltre i numeri casuali sono generati a partire da 16 bit del registro
 - 2.3 In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell' attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)
3. L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM⁺08]
4. La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari
5. LFSR State Recovery
6. LFSR Rollback
7. I bit di parità sono computati sul plaintext e poi inviati non cifrati

Vulnerabilità

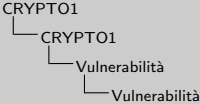
1. Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]
2. RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM⁺08] con condizione iniziale costante.

2.1 Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM⁺08][CNO08]

2.2 Inoltre i numeri casuali sono generati a partire da 16 bit del registro

2.3 In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell' attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)
3. L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM⁺08]
4. La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari
5. LFSR State Recovery
6. LFSR Rollback
7. I bit di parità sono computati sul plaintext e poi inviati non cifrati

2022-03-24



Vulnerabilità

1. Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]
2. RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM⁺08] con condizione iniziale costante.

2.1 Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM⁺08][CNO08]

2.2 Inoltre i numeri casuali sono generati a partire da 16 bit del registro

2.3 In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell' attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)
3. L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM⁺08]
4. La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari
5. LFSR State Recovery
6. LFSR Rollback
7. I bit di parità sono computati sul plaintext e poi inviati non cifrati

1	Sommario
2	ISO14443
3	Cifrari di flusso
4	CRYPTO1
	Storia
	Vulnerabilità
	Reverse Engineering
	Attacchi
	Considerazioni
	Successori
5	Bibliografia

1. Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]

2. RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM⁺08] con condizione iniziale costante.

2.1 Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM⁺08][CNO08]

2.2 Inoltre i numeri casuali sono generati a partire da 16 bit del registro

2.3 In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell' attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)

3. L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM⁺08]

4. La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari

5. LFSR State Recovery

6. LFSR Rollback

7. I bit di parità sono computati sul plaintext e poi inviati non cifrati

2022-03-24

CRYPTO1

CRYPTO1

Vulnerabilità

Vulnerabilità

Vulnerabilità

1. Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]

2. RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM⁺08] con condizione iniziale costante.

2.1 Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM⁺08][CNO08]

2.2 Inoltre i numeri casuali sono generati a partire da 16 bit del registro

2.3 In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell' attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)

3. L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM⁺08]

4. La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari

5. LFSR State Recovery

6. LFSR Rollback

7. I bit di parità sono computati sul plaintext e poi inviati non cifrati

21

Vulnerabilità

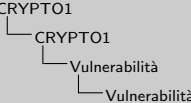
1. Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]
2. RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM⁺08] con condizione iniziale costante.

2.1 Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM⁺08][CNO08]

2.2 Inoltre i numeri casuali sono generati a partire da 16 bit del registro

2.3 In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell' attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)
3. L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM⁺08]
4. La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari
5. LFSR State Recovery
6. LFSR Rollback
7. I bit di parità sono computati sul plaintext e poi inviati non cifrati

2022-03-24



Vulnerabilità

1. Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]
2. RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM⁺08] con condizione iniziale costante.

2.1 Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM⁺08][CNO08]

2.2 Inoltre i numeri casuali sono generati a partire da 16 bit del registro

2.3 In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell' attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)
3. L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM⁺08]
4. La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari
5. LFSR State Recovery
6. LFSR Rollback
7. I bit di parità sono computati sul plaintext e poi inviati non cifrati

1	Sommario
2	ISO14443
3	Cifrari di flusso
4	CRYPTO1
	Storia
	Vulnerabilità
	Reverse Engineering
	Attacchi
	Considerazioni
	Successori
5	Bibliografia

1. Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]

2. RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM⁺08] con condizione iniziale costante.

2.1

Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM⁺08][CNO08]

2.2

Inoltre i numeri casuali sono generati a partire da 16 bit del registro

2.3

In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell' attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)

3. L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM⁺08]

4. La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari

5. LFSR State Recovery

6. LFSR Rollback

7. I bit di parità sono computati sul plaintext e poi inviati non cifrati

2022-03-24

CRYPTO1

CRYPTO1

Vulnerabilità

Vulnerabilità

Vulnerabilità

1. Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]

2. RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM⁺08] con condizione iniziale costante.

2.1

Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM⁺08][CNO08]

2.2

Inoltre i numeri casuali sono generati a partire da 16 bit del registro

2.3

In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell' attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)

3. L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM⁺08]

4. La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari

5. LFSR State Recovery

6. LFSR Rollback

7. I bit di parità sono computati sul plaintext e poi inviati non cifrati

1	Sommario
2	ISO14443
3	Cifrari di flusso
4	CRYPTO1
	Storia
	Vulnerabilità
	Reverse Engineering
	Attacchi
	Considerazioni
	Successori
5	Bibliografia

1. Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]

2. RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM⁺08] con condizione iniziale costante.

2.1

Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM⁺08][CNO08]

2.2

Inoltre i numeri casuali sono generati a partire da 16 bit del registro

2.3

In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell' attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)

3. L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM⁺08]

4. La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari

5. LFSR State Recovery

6. LFSR Rollback

7. I bit di parità sono computati sul plaintext e poi inviati non cifrati

2022-03-24

CRYPTO1

CRYPTO1

Vulnerabilità

Vulnerabilità

Vulnerabilità

1. Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]

2. RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM⁺08] con condizione iniziale costante.

2.1

Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM⁺08][CNO08]

2.2

Inoltre i numeri casuali sono generati a partire da 16 bit del registro

2.3

In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell' attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)

3. L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM⁺08]

4. La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari

5. LFSR State Recovery

6. LFSR Rollback

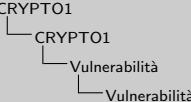
7. I bit di parità sono computati sul plaintext e poi inviati non cifrati

1	Sommario
2	ISO14443
3	Cifrari di flusso
4	CRYPTO1
5	Bibliografia
6	Storia
7	Vulnerabilità
8	Reverse Engineering
9	Attacchi
10	Considerazioni
11	Successori
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	
41	
42	
43	
44	
45	
46	
47	
48	
49	
50	
51	
52	
53	
54	
55	
56	
57	
58	
59	
60	
61	
62	
63	
64	
65	
66	
67	
68	
69	
70	
71	
72	
73	
74	
75	
76	
77	
78	
79	
80	
81	
82	
83	
84	
85	
86	
87	
88	
89	
90	
91	
92	
93	
94	
95	
96	
97	
98	
99	
100	
101	
102	
103	
104	
105	
106	
107	
108	
109	
110	
111	
112	
113	
114	
115	
116	
117	
118	
119	
120	
121	
122	
123	
124	
125	
126	
127	
128	
129	
130	
131	
132	
133	
134	
135	
136	
137	
138	
139	
140	
141	
142	
143	
144	
145	
146	
147	
148	
149	
150	
151	
152	
153	
154	
155	
156	
157	
158	
159	
160	
161	
162	
163	
164	
165	
166	
167	
168	
169	
170	
171	
172	
173	
174	
175	
176	
177	
178	
179	
180	
181	
182	
183	
184	
185	
186	
187	
188	
189	
190	
191	
192	
193	
194	
195	
196	
197	
198	
199	
200	
201	
202	
203	
204	
205	
206	
207	
208	
209	
210	
211	
212	
213	
214	
215	
216	
217	
218	
219	
220	
221	
222	
223	
224	
225	
226	
227	
228	
229	
230	
231	
232	
233	
234	
235	
236	
237	
238	
239	
240	
241	
242	
243	
244	
245	
246	
247	
248	
249	
250	
251	
252	
253	
254	
255	
256	
257	
258	
259	
260	
261	
262	
263	
264	
265	
266	
267	
268	
269	
270	
271	
272	
273	
274	
275	
276	
277	
278	
279	
280	
281	
282	
283	
284	
285	
286	
287	
288	
289	
290	
291	
292	
293	
294	
295	
296	
297	
298	
299	
300	
301	
302	
303	
304	
305	
306	
307	
308	
309	
310	
311	
312	
313	
314	
315	
316	
317	
318	
319	
320	
321	
322	
323	
324	
325	
326	
327	
328	
329	
330	
331	
332	
333	
334	
335	
336	
337	
338	
339	
340	
341	
342	
343	
344	
345	
346	
347	
348	
349	
350	
351	
352	
353	
354	
355	
356	
357	
358	
359	
360	
361	
362	
363	
364	
365	
366	
367	
368	
369	
370	
371	
372	
373	
374	
375	
376	
377	
378	
379	
380	
381	
382	
383	
384	
385	
386	
387	
388	
389	
390	
391	
392	
393	
394	
395	
396	
397	
398	
399	
400	
401	
402	
403	
404	
405	
406	
407	
408	
409	
410	
411	
412	
413	
414	
415	
416	
417	
418	
419	
420	
421	
422	
423	
424	
425	
426	
427	
428	
429	
430	
431	
432	
433	
434	
435	
436	
437	
438	
439	
440	
441	
442	
443	
444	
445	
446	
447	
448	
449	
450	
451	
452	
453	
454	
455	
456	
457	
458	
459	
460	
461	
462	
463	
464	
465	
466	
467	
468	
469	
470	
471	
472	
473	
474	
475	
476	
477	
478	
479	
480	
481	
482	
483	
484	
485	
486	
487	
488	
489	
490	
491	
492	
493	
494	
495	
496	
497	
498	
499	
500	
501	
502	
503	
504	
505	
506	
507	
508	
509	
510	
511	
512	
513	
514	
515	
516	
517	
518	
519	
520	
521	
522	
523	
524	
525	
526	
527	
528	
529	
530	
531	
532	
533	
534	
535	
536	
537	
538	
539	
540	
541	
542	
543	
544	
545	
546	
547	
548	
549	
550	
551	
552	
553	
554	
555	
556	
557	
558	
559	
560	
561	
562	
563	
564	
565	
566	
567	
568	
569	
570	
571	
572	
573	
574	
575	
576	
577	
578	
579	
580	
581	
582	
583	
584	
585	
586	
587	
588	
589	
590	
591	
592	
593	
594	
595	
596	
597	
598	
599	
600	
601	
602	
603	
604	
605	
606	
607	
608	
609	
610	
611	
612	
613	
614	
615	
616	
617	
618	
619	
620	
621	
622	
623	
624	
625	
626	
627	
628	
629	
630	
631	
632	
633	
634	
635	
636	
637	
638	
639	
640	
641	
642	
643	
644	
645	
646	
647	
648	
649	
650	
651	
652	
653	
654	
655	
656	
657	
658	
659	
660	
661	
662	
663	
664	
665	
666	
667	
668	
669	
670	
671	
672	
673	
674	
675	
676	
677	
678	
679	
680	
681	
682	
683	
684	
685	
686	
687	
688	
689	
690	
691	
69	

Vulnerabilità

1. Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]
2. RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM⁺08] con condizione iniziale costante.
 - 2.1 Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM⁺08][CNO08]
 - 2.2 Inoltre i numeri casuali sono generati a partire da 16 bit del registro
 - 2.3 In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell' attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)
3. L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM⁺08]
4. La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari
5. LFSR State Recovery
6. LFSR Rollback
7. I bit di parità sono computati sul plaintext e poi inviati non cifrati

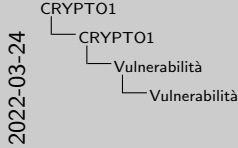
2022-03-24



Vulnerabilità

1. Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]
2. RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM⁺08] con condizione iniziale costante.
 - 2.1 Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM⁺08][CNO08]
 - 2.2 Inoltre i numeri casuali sono generati a partire da 16 bit del registro
 - 2.3 In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell' attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)
3. L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM⁺08]
4. La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari
5. LFSR State Recovery
6. LFSR Rollback
7. I bit di parità sono computati sul plaintext e poi inviati non cifrati

1	Sommario
2	ISO14443
3	Cifrari di flusso
4	CRYPTO1
5	Bibliografia
6	Vulnerabilità
7	Reverse Engineering
8	Attacchi
9	Considerazioni
10	Successori
11	21



1.	Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]
2.	RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM ⁺ 08] con condizione iniziale costante.
2.1	Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM ⁺ 08][CNO08]
2.2	Inoltre i numeri casuali sono generati a partire da 16 bit del registro
2.3	In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell'attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)
3.	L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM ⁺ 08]
4.	La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari
5.	LFSR State Recovery
6.	LFSR Rollback
7.	I bit di parità sono computati sul plaintext e poi inviati non cifrati

Vulnerabilità

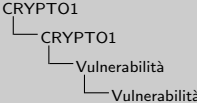
1. Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]
2. RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM⁺08] con condizione iniziale costante.

2.1 Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM⁺08][CNO08]

2.2 Inoltre i numeri casuali sono generati a partire da 16 bit del registro

2.3 In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell' attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)
3. L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM⁺08]
4. La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari
5. LFSR State Recovery
6. LFSR Rollback
7. I bit di parità sono computati sul plaintext e poi inviati non cifrati

2022-03-24



- 1 L'unica protezione contro attacchi bruteforce sta nel tempo di transazione di alcuni millisecondi ma non c'è nessuna protezione da attacchi su ciphertext.
- 2 Dalla analisi dei pacchetti è stato possibile notare che lo stato del RNG è di soli 16 bit. Questo comporta l'esistenza di soli 65536 possibili nonce e che la seconda parte di n_t sia funzione della prima. In particolare è stato possibile identificare che un nonce è valido se e solo se è rispettato

$$n_k \oplus n_k + 2 \oplus n_k + 3 \oplus n_k + 5 \oplus n_k + 16 = 0$$

- 2.1 Tramite tentativi e analisi dei dati trasmessi è stato possibile rilevare che il valore del nonce n_t dipende solamente dal tempo di accensione del tag. Questo permette di evincere che il generatore di numeri casuali ha un seed iniziale codificato nell'hardware.

- 3 Questo implica che possono esistere numerosi lettori vulnerabili ad attacchi sul parametro n_r , infatti è possibile notare che la sequenza dei nonce n_r a partire dall'avvio del lettore non varia.

Vulnerabilità

1. Utilizzo di chiavi a 48 bit: Possibili attacchi BruteForce [CNO08]
2. RNG del tag non è crittograficamente sicuro. Infatti è un LFSR (a 16 bit) [GKGM⁺08] con condizione iniziale costante.

2.1 Consegue che lo stato è prevedibile e dipende dal tempo trascorso dal power-on [GKGM⁺08][CNO08]

2.2 Inoltre i numeri casuali sono generati a partire da 16 bit del registro

2.3 In particolare i numeri sono generati ad ogni ciclo di clock del tag, quindi la precisione dell' attaccante deve limitarsi a quanti di 10 microsecondi (106kHz) e la sequenza di numeri si ripete ogni 65535 iterazioni (0.6s)
3. L'RNG dei lettori viene aggiornato solamente ad ogni nuova autenticazione [GKGM⁺08]
4. La funzione di filtraggio del LFSR usa 20bit del registro e sono solo bit in posizione dispari
5. LFSR State Recovery
6. LFSR Rollback
7. I bit di parità sono computati sul plaintext e poi inviati non cifrati

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse
Engineering

Attacchi

Considerazioni

Successori

5 Bibliogra-
fia

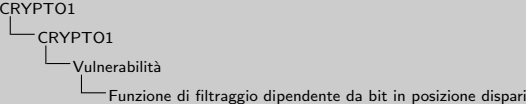
Funzione di filtraggio dipendente da bit in posizione dispari

La funzione di filtraggio è stata identificata ed è stato notato come solo i bit in posizione dispari siano input ad essa.

I bit dispari dello stato sono responsabili della codifica dei bit dispari del messaggio

I bit pari dello stato sono responsabili della codifica dei bit pari del messaggio

2022-03-24



Funzione di filtraggio dipendente da bit in posizione dispari

La funzione di filtraggio è stata identificata ed è stato notato come solo i bit in posizione dispari siano input ad essa.

I bit dispari dello stato sono responsabili della codifica dei bit dispari del messaggio

I bit pari dello stato sono responsabili della codifica dei bit pari del messaggio

Funzione di filtraggio dipendente da bit in posizione dispari

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

- Storia
- Vulnerabilità
- Reverse Engineering
- Attacchi
- Considerazioni
- Successori

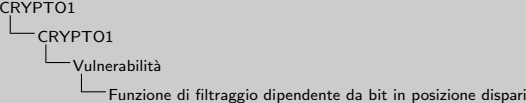
5 Bibliogra-
fia

La funzione di filtraggio è stata identificata ed è stato notato come solo i bit in posizione dispari siano input ad essa.

I bit dispari dello stato sono responsabili della codifica dei bit dispari del messaggio

I bit pari dello stato sono responsabili della codifica dei bit pari del messaggio

2022-03-24

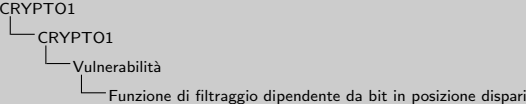


Funzione di filtraggio dipendente da bit in posizione dispari

La funzione di filtraggio è stata identificata ed è stato notato come solo i bit in posizione dispari siano input ad essa.
I bit dispari dello stato sono responsabili della codifica dei bit dispari del messaggio
I bit pari dello stato sono responsabili della codifica dei bit pari del messaggio

Funzione di filtraggio dipendente da bit in posizione dispari

2022-03-24



Funzione di filtraggio dipendente da bit in posizione dispari

La funzione di filtraggio è stata identificata ed è stato notato come solo i bit in posizione dispari siano input ad essa.
I bit dispari dello stato sono responsabili della codifica dei bit dispari del messaggio
I bit pari dello stato sono responsabili della codifica dei bit pari del messaggio

È così possibile ricostruire lo stato a partire da un keystream dividendo l'identificazione in due processi paralleli, uno che permetterà di trovare i bit pari e uno i bit dispari.

Per precisione, siano $b_0, b_1, b_2 \dots b_{n-1}$ i bit del keystream (per un numero pari di bit, dato che i messaggi hanno sempre lunghezza pari). Bisogna trovare la sequenza $\bar{r} = r_0, r_1, r_2 \dots r_{46+n}$ tale per cui valga la relazione

$$\begin{aligned} & r_k \oplus r_k + 5 \oplus r_k + 9 \oplus r_k + 10 \oplus r_k + 12 \oplus r_k + 14 \oplus r_k + 15 \oplus r_k + 17 \\ & \oplus r_k + 19 \oplus r_k + 24 \oplus r_k + 25 \oplus r_k + 27 \oplus r_k + 29 \oplus r_k + 35 \oplus r_k + 39 \oplus r_k + 41 \\ & \oplus r_k + 42 \oplus r_k + 43 \oplus r_k + 48 = 0 \end{aligned}$$

$$\forall k \in \{0, \dots, n - 2\}$$

e tale che

$$f(r_k \dots r_k + 47) = b_k, \forall k \in \{0, \dots, n - 1\}.$$

In questo modo tutte le sotto sequenze di \bar{r} di lunghezza 48 saranno stati in successione del LFSR

Bit di parità

Ulteriore vulnerabilità dell'implementazione è il fatto che il livello di trasmissione fisica richiede un invio di un bit di parità ogni 8 bit di dato

IL BIT DI PARITÀ È QUINDI INVIATO CIFRATO **CONDIVIDENDO IL BIT DEL KEYSTREAM** MA VIENE **CALCOLATO SUL PLAINTEXT**[Cou09]

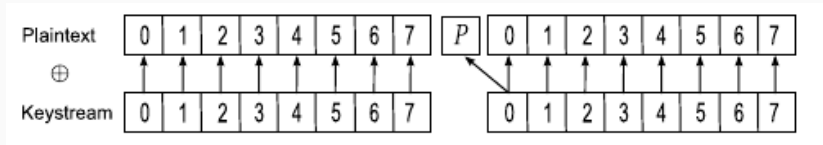
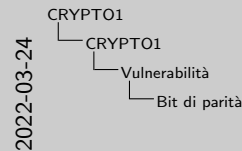


Figura 7: Assegnazione dei bit di cifratura tra dati e bit di parità



Questo permette di avere informazioni sul keystream (almeno su un bit ogni 8) ed è una proprietà utilizzabile per ridurre lo spazio di ricerca negli

attacchi Nested authentication (vedi slide 42)



Errori di trasmissione

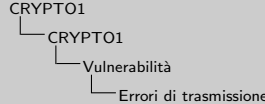
Secondo il protocollo ISO14443a ogni 8 bit di dati deve essere inviato un bit di parità.

Il tag riceve una sequenza di dati con bit di parità invalidi \implies Nessuna risposta da parte del tag

Il tag riceve una sequenza con bit di parità corretti ma la verifica della risposta alla challenge non ha successo \implies NACK

IL NACK È CIFRATO MA È UN TESTO CONOSCIUTO

2022-03-24



Errori di trasmissione

Secondo il protocollo ISO14443a ogni 8 bit di dati deve essere inviato un bit di parità.

Il tag riceve una sequenza di dati con bit di parità invalidi \implies Nessuna risposta da parte del tag

Il tag riceve una sequenza con bit di parità corretti ma la verifica della risposta alla challenge non ha successo \implies NACK

IL NACK È CIFRATO MA È UN TESTO CONOSCIUTO

Errori di trasmissione

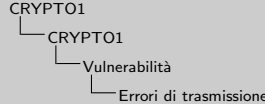
Secondo il protocollo ISO14443a ogni 8 bit di dati deve essere inviato un bit di parità.

Il tag riceve una sequenza di dati con bit di parità invalidi \implies Nessuna risposta da parte del tag

Il tag riceve una sequenza con bit di parità corretti ma la verifica della risposta alla challenge non ha successo \implies NACK

IL NACK È CIFRATO MA È UN TESTO CONOSCIUTO

2022-03-24



Errori di trasmissione

Secondo il protocollo ISO14443a ogni 8 bit di dati deve essere inviato un bit di parità.

Il tag riceve una sequenza di dati con bit di parità invalidi \implies Nessuna risposta da parte del tag

Il tag riceve una sequenza con bit di parità corretti ma la verifica della risposta alla challenge non ha successo \implies NACK

IL NACK È CIFRATO MA È UN TESTO CONOSCIUTO

Errori di trasmissione

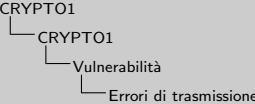
Secondo il protocollo ISO14443a ogni 8 bit di dati deve essere inviato un bit di parità.

Il tag riceve una sequenza di dati con bit di parità invalidi \implies Nessuna risposta da parte del tag

Il tag riceve una sequenza con bit di parità corretti ma la verifica della risposta alla challenge non ha successo \implies NACK

IL NACK È CIFRATO MA È UN TESTO CONOSCIUTO

2022-03-24



Errori di trasmissione

Secondo il protocollo ISO14443a ogni 8 bit di dati deve essere inviato un bit di parità.

Il tag riceve una sequenza di dati con bit di parità invalidi \implies Nessuna risposta da parte del tag

Il tag riceve una sequenza con bit di parità corretti ma la verifica della risposta alla challenge non ha successo \implies NACK

IL NACK È CIFRATO MA È UN TESTO CONOSCIUTO

Errori di trasmissione

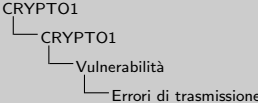
Secondo il protocollo ISO14443a ogni 8 bit di dati deve essere inviato un bit di parità.

Il tag riceve una sequenza di dati con bit di parità invalidi \implies Nessuna risposta da parte del tag

Il tag riceve una sequenza con bit di parità corretti ma la verifica della risposta alla challenge non ha successo \implies NACK

IL NACK È CIFRATO MA È UN TESTO CONOSCIUTO

2022-03-24



Questo permette, dopo aver collezionato un buon quantitativo di NACK, di attaccare il cifrario tramite inversione della funzione di filtraggio e il successivo rollback.

Errori di trasmissione

Secondo il protocollo ISO14443a ogni 8 bit di dati deve essere inviato un bit di parità.
Il tag riceve una sequenza di dati con bit di parità invalidi \implies Nessuna risposta da parte del tag
Il tag riceve una sequenza con bit di parità corretti ma la verifica della risposta alla challenge non ha successo \implies NACK
IL NACK È CIFRATO MA È UN TESTO CONOSCIUTO

Reverse engineering

Uno dei punti di forza di MIFARE è il bassissimo costo di produzione.

Il circuito che si occupa della logica dell'integrato è composto da circa 400 porte NAND (equivalenti) [NESP08]

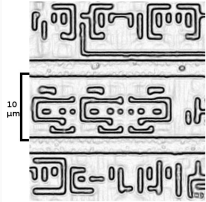
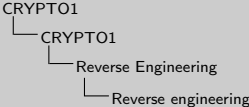


Figura 8: Immagine tratta da [NESP08] presa da un microscopio ottico (500x) e processata con filtri di edge detection

2022-03-24



Reverse engineering

Uno dei punti di forza di MIFARE è il bassissimo costo di produzione.
Il circuito che si occupa della logica dell'integrato è composto da circa 400 porte NAND (equivalenti) [NESP08]



Figura 8: Immagine tratta da [NESP08] presa da un microscopio ottico (500x) e processata con filtri di edge detection

- 1 Sommario
- 2 ISO14443
- 3 Cifrari di flusso
- 4 CRYPTO1
 - Storia
 - Vulnerabilità
 - Reverse Engineering
 - Filter Function
 - Attacchi
 - Considerazioni
 - Successori
- 5 Bibliografia

Reverse engineering

Grazie alle ricerche e allo studio degli integrati (fig 8), è stato possibile dedurre il circuito logico utilizzato al fine di implementare la crittografia della fase di autenticazione.

Il circuito si presenta come segue:

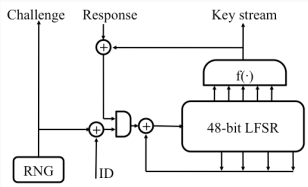
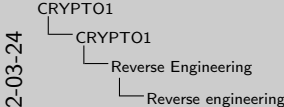


Figura 9: Circuito logico ricavato dal reverse engineering del tag



2022-03-24

Reverse engineering

Grazie alle ricerche e allo studio degli integrati (fig 8), è stato possibile dedurre il circuito logico utilizzato al fine di implementare la crittografia della fase di autenticazione.

Il circuito si presenta come segue:



Figura 9: Circuito logico ricavato dal reverse engineering del tag

Dal seguente schema è possibile ricavare il funzionamento completo del processo di autenticazione a un blocco. In particolare è possibile definire con certezza le funzioni

$$cipherInit(key, uid, n_t) = f(key)$$

Si noti che la funzione configura i parametri uid e n_t proveniente dal RNG.

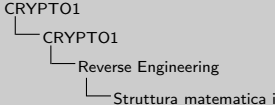
$$cipher(n_r) = f(next_iteration())$$

dove n_r viene impostato precedentemente all'iterazione.

Il procedimento permette così di configurare in modo identico i due stream cipher per garantire di avere due stream di chiavi uguali per cifrare la comunicazione.

Struttura matematica i

2022-03-24



1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse
Engineering

Filter Function

Attacchi

Considerazioni

Successori

5 Bibliogra-
fia

Indichiamo var_i l'i-esimo bit della variabile var

Indichiamo $bits(num)$ la funzione che ritorna il numero di bit che compongono num

Indichiamo $L(LFSR)$ il risultato della funzione di feedback del LFSR [GKGM⁺08]

Definiamo la funzione $set(key, s)$ che imposta il valore di LFSR allo stato s con il
valore key

$$set(key, s) = LFSR_i := key_i \forall i \in [0, bits(num))$$

Definiamo quindi la funzione $next(bit, s)$ la funzione che effettua l'iterazione sullo
LFSR

$$next(bit, s) = set(LFSR[47..1] || L(LFSR) \oplus bit, s)$$

Struttura matematica i

Indichiamo var_i l'i-esimo bit della variabile var
Indichiamo $bits(num)$ la funzione che ritorna il numero di bit che compongono num
Indichiamo $L(LFSR)$ il risultato della funzione di feedback del LFSR [GKGM⁺08]

Definiamo la funzione $set(key, s)$ che imposta il valore di LFSR allo stato s con il
valore key

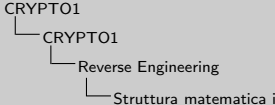
$$set(key, s) = LFSR_i := key[i] \forall i \in [0, bits(num))$$

Definiamo quindi la funzione $next(bit, s)$ la funzione che effettua l'iterazione sullo
LFSR

$$next(bit, s) = set(LFSR[47..1] || L(LFSR) \oplus bit, s)$$

Struttura matematica i

2022-03-24



1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse
Engineering

Filter Function

Attacchi

Considerazioni

Successori

5 Bibliogra-
fia

Indichiamo var_i l'i-esimo bit della variabile var

Indichiamo $bits(num)$ la funzione che ritorna il numero di bit che compongono num

Indichiamo $L(LFSR)$ il risultato della funzione di feedback del LFSR [GKGM⁺08]

Definiamo la funzione $set(key, s)$ che imposta il valore di LFSR allo stato s con il valore key

$$set(key, s) = LFSR_i := key_i \forall i \in [0, bits(num))$$

Definiamo quindi la funzione $next(bit, s)$ la funzione che effettua l'iterazione sullo LFSR

$$next(bit, s) = set(LFSR[47..1] || L(LFSR) \oplus bit, s)$$

Struttura matematica i

Indichiamo var_i l'i-esimo bit della variabile var
Indichiamo $bits(num)$ la funzione che ritorna il numero di bit che compongono num
Indichiamo $L(LFSR)$ il risultato della funzione di feedback del LFSR [GKGM⁺08]
Definiamo la funzione $set(key, s)$ che imposta il valore di LFSR allo stato s con il valore key

$$set(key, s) = LFSR_i := key_i \forall i \in [0, bits(num))$$

Definiamo quindi la funzione $next(bit, s)$ la funzione che effettua l'iterazione sullo LFSR

$$next(bit, s) = set(LFSR[47..1] || L(LFSR) \oplus bit, s)$$

Struttura matematica i

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse
Engineering

Filter Function

Attacchi

Considerazioni

Successori

5 Bibliogra-
fia

Indichiamo var_i l'i-esimo bit della variabile var

Indichiamo $bits(num)$ la funzione che ritorna il numero di bit che compongono num

Indichiamo $L(LFSR)$ il risultato della funzione di feedback del LFSR [GKGM⁺08]

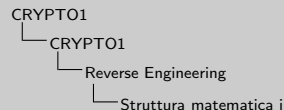
Definiamo la funzione $set(key, s)$ che imposta il valore di LFSR allo stato s con il
valore key

$$set(key, s) = LFSR_i := key_i \forall i \in [0, bits(num))$$

Definiamo quindi la funzione $next(bit, s)$ la funzione che effettua l'iterazione sullo
LFSR

$$next(bit, s) = set(LFSR[47..1] || L(LFSR) \oplus bit, s)$$

2022-03-24



Struttura matematica i

Indichiamo var_i l'i-esimo bit della variabile var
Indichiamo $bits(num)$ la funzione che ritorna il numero di bit che compongono num
Indichiamo $L(LFSR)$ il risultato della funzione di feedback del LFSR [GKGM⁺08]
Definiamo la funzione $set(key, s)$ che imposta il valore di LFSR allo stato s con il
valore key

$$set(key, s) = LFSR_i := key_i \forall i \in [0, bits(num))$$

Definiamo quindi la funzione $next(bit, s)$ la funzione che effettua l'iterazione sullo
LFSR

$$next(bit, s) = set(LFSR[47..1] || L(LFSR) \oplus bit, s)$$

Struttura matematica ii

Indichiamo la funzione $f(LFSR, s)$ la filter function (si veda [GKGM⁺08]) che ritorna il valore del keystream allo stato s

Indichiamo la funzione $rnd()$ la funzione che genera un numero casuale

Definiamo $push(value, s)$ la funzione che aggiorna lo LFSR con il valore $value$

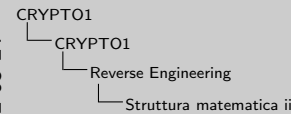
$$push(value, s) = next(value_i, s) \forall i \in [0, bits(value))$$

Indichiamo $suc^n(seed)$ la funzione che ottiene i successivi 32 bit dal RNG [Hua12]

Definiamo infine $ks(s)$ la funzione che ritorna la chiave corrispondente allo stato s

$$ks(s) = f(LFSR, s)$$

2022-03-24



Struttura matematica ii

Indichiamo la funzione $f(LFSR, s)$ la filter function (si veda [GKGM⁺08]) che ritorna il valore del keystream allo stato s

Indichiamo la funzione $rnd()$ la funzione che genera un numero casuale

Definiamo $push(value, s)$ la funzione che aggiorna lo LFSR con il valore $value$

$push(value, s) = next(value_i, s) \forall i \in [0, bits(value))$

Indichiamo $suc^n(seed)$ la funzione che ottiene i successivi 32 bit dal RNG [Hua12]

Definiamo infine $ks(s)$ la funzione che ritorna la chiave corrispondente allo stato s

$ks(s) = f(LFSR, s)$

1 Sommario

2 ISO14443

3 Cifrari di flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse Engineering

Filter Function

Attacchi

Considerazioni

Successori

5 Bibliografia

Struttura matematica ii

Indichiamo la funzione $f(LFSR, s)$ la filter function (si veda [GKGM⁺08]) che ritorna il valore del keystream allo stato s

Indichiamo la funzione $rnd()$ la funzione che genera un numero casuale

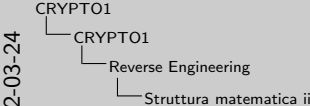
Definiamo $push(value, s)$ la funzione che aggiorna lo LFSR con il valore $value$

$$push(value, s) = next(value_i, s) \forall i \in [0, bits(value))$$

Indichiamo $suc^n(seed)$ la funzione che ottiene i successivi 32 bit dal RNG [Hua12]

Definiamo infine $ks(s)$ la funzione che ritorna la chiave corrispondente allo stato s

$$ks(s) = f(LFSR, s)$$



Struttura matematica ii

Indichiamo la funzione $f(LFSR, s)$ la filter function (si veda [GKGM⁺08]) che ritorna il valore del keystream allo stato s

Indichiamo la funzione $rnd()$ la funzione che genera un numero casuale

Definiamo $push(value, s)$ la funzione che aggiorna lo LFSR con il valore $value$

$$push(value, s) = next(value_i, s) \forall i \in [0, bits(value))$$

Indichiamo $suc^n(seed)$ la funzione che ottiene i successivi 32 bit dal RNG [Hua12]

Definiamo infine $ks(s)$ la funzione che ritorna la chiave corrispondente allo stato s

$$ks(s) = f(LFSR, s)$$

Struttura matematica ii

Indichiamo la funzione $f(LFSR, s)$ la filter function (si veda [GKGM⁺08]) che ritorna il valore del keystream allo stato s

Indichiamo la funzione $rnd()$ la funzione che genera un numero casuale

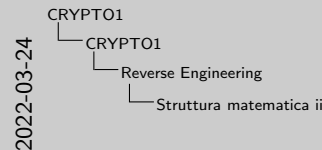
Definiamo $push(value, s)$ la funzione che aggiorna lo LFSR con il valore $value$

$$push(value, s) = next(value_i, s) \forall i \in [0, bits(value))$$

Indichiamo $suc^n(seed)$ la funzione che ottiene i successivi 32 bit dal RNG [Hua12]

Definiamo infine $ks(s)$ la funzione che ritorna la chiave corrispondente allo stato s

$$ks(s) = f(LFSR, s)$$



Struttura matematica ii

Indichiamo la funzione $f(LFSR, s)$ la filter function (si veda [GKGM⁺08]) che ritorna il valore del keystream allo stato s

Indichiamo la funzione $rnd()$ la funzione che genera un numero casuale

Definiamo $push(value, s)$ la funzione che aggiorna lo LFSR con il valore $value$

$$push(value, s) = next(value_i, s) \forall i \in [0, bits(value))$$

Indichiamo $suc^n(seed)$ la funzione che ottiene i successivi 32 bit dal RNG [Hua12]

Definiamo infine $ks(s)$ la funzione che ritorna la chiave corrispondente allo stato s

$$ks(s) = f(LFSR, s)$$

1 Sommario

2 ISO14443

3 Cifrari di flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse Engineering

Filter Function

Attacchi

Considerazioni

Successori

5 Bibliografia

Struttura matematica ii

Indichiamo la funzione $f(LFSR, s)$ la filter function (si veda [GKGM⁺08]) che ritorna il valore del keystream allo stato s

Indichiamo la funzione $rnd()$ la funzione che genera un numero casuale

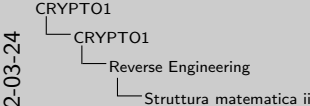
Definiamo $push(value, s)$ la funzione che aggiorna lo LFSR con il valore $value$

$$push(value, s) = next(value_i, s) \forall i \in [0, bits(value))$$

Indichiamo $suc^n(seed)$ la funzione che ottiene i successivi 32 bit dal RNG [Hua12]

Definiamo infine $ks(s)$ la funzione che ritorna la chiave corrispondente allo stato s

$$ks(s) = f(LFSR, s)$$



Struttura matematica ii
Indichiamo la funzione $f(LFSR, s)$ la filter function (si veda [GKGM ⁺ 08]) che ritorna il valore del keystream allo stato s
Indichiamo la funzione $rnd()$ la funzione che genera un numero casuale
Definiamo $push(value, s)$ la funzione che aggiorna lo LFSR con il valore $value$
$push(value, s) = next(value_i, s) \forall i \in [0, bits(value))$
Indichiamo $suc^n(seed)$ la funzione che ottiene i successivi 32 bit dal RNG [Hua12]
Definiamo infine $ks(s)$ la funzione che ritorna la chiave corrispondente allo stato s
$ks(s) = f(LFSR, s)$

Struttura matematica iii

1

Sommario

2

ISO14443

3

Cifrari di flusso

4

CRYPTO1

Storia

Vulnerabilità

Reverse Engineering

Filter Function

Attacchi

Considerazioni

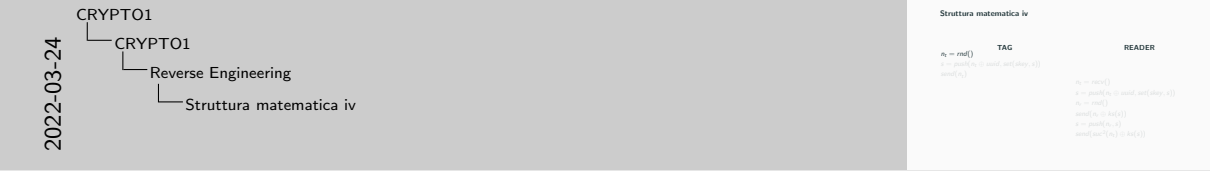
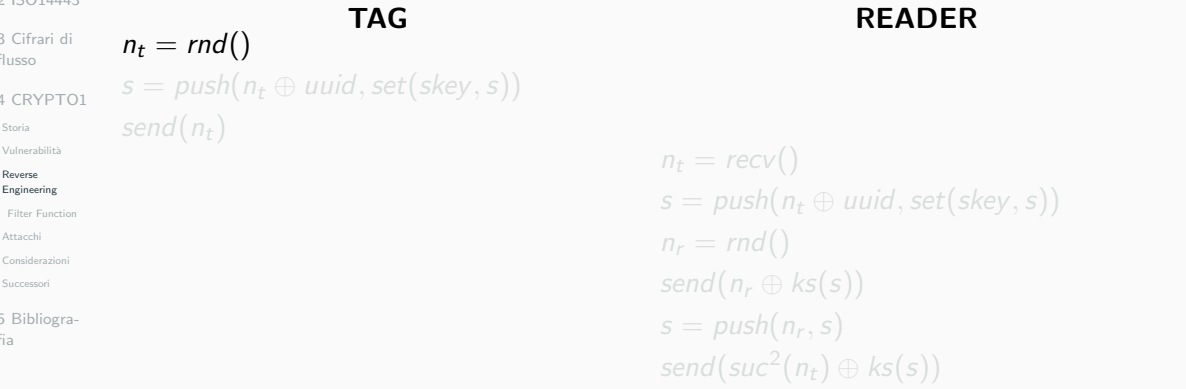
Successori

5

Bibliografia



1	Sommario
2	ISO14443
3	Cifrari di flusso
4	CRYPTO1
	Storia
	Vulnerabilità
	Reverse Engineering
	Filter Function
	Attacchi
	Considerazioni
	Successori
5	Bibliografia



Struttura matematica iv		
	TAG	READER
	$n_t = rnd()$ $s = push(n_t \oplus uuid, set(skey, s))$ $send(n_t)$	$n_t = recv()$ $s = push(n_t \oplus uuid, set(skey, s))$ $n_r = rnd()$ $send(n_r \oplus ks(s))$ $s = push(n_r, s)$ $send(suc^2(n_t) \oplus ks(s))$

1	Sommario
2	ISO14443
3	Cifrari di flusso
4	CRYPTO1
	Storia
	Vulnerabilità
	Reverse Engineering
	Filter Function
	Attacchi
	Considerazioni
	Successori
5	Bibliografia

2022-03-24

CRYPTO1

CRYPTO1

Reverse Engineering

Struttura matematica iv

Struttura matematica iv

TAG

READER

	TAG	READER
	$n_t = rnd()$	
	$s = push(n_t \oplus uuid, set(skey, s))$	
	$send(n_t)$	
		$n_t = recv()$
		$s = push(n_t \oplus uuid, set(skey, s))$
		$n_r = rnd()$
		$send(n_r \oplus ks(s))$
		$s = push(n_r, s)$
		$send(suc^2(n_t) \oplus ks(s))$

TAG	READER
$n_t = rnd()$ $s = push(n_t \oplus uuid, set(skey, s))$ $send(n_t)$	$n_t = recv()$ $s = push(n_t \oplus uuid, set(skey, s))$ $n_r = rnd()$ $send(n_r \oplus ks(s))$ $s = push(n_r, s)$ $send(suc^2(n_t) \oplus ks(s))$

Struttura matematica iv

TAG

$$n_t = rnd()$$

$$s = push(n_t \oplus uuid, set(skey, s))$$

$$send(n_t)$$

READER

$$n_t = recv()$$

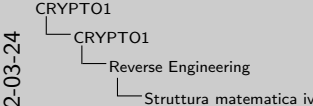
$$s = push(n_t \oplus uuid, set(skey, s))$$

$$n_r = rnd()$$

$$send(n_r \oplus ks(s))$$

$$s = push(n_r, s)$$

$$send(suc^2(n_t) \oplus ks(s))$$



TAG
 $n_t = rnd()$
 $s = push(n_t \oplus uuid, set(skey, s))$
 $send(n_t)$

READER
 $n_t = recv()$
 $s = push(n_t \oplus uuid, set(skey, s))$
 $n_r = rnd()$
 $send(n_r \oplus ks(s))$
 $s = push(n_r, s)$
 $send(suc^2(n_t) \oplus ks(s))$

Struttura matematica iv

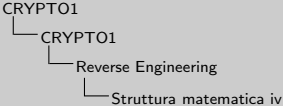
TAG

$$n_t = rnd()$$
$$s = push(n_t \oplus uuid, set(skey, s))$$
$$send(n_t)$$

READER

$$n_t = recv()$$
$$s = push(n_t \oplus uuid, set(skey, s))$$
$$n_r = rnd()$$
$$send(n_r \oplus ks(s))$$
$$s = push(n_r, s)$$
$$send(suc^2(n_t) \oplus ks(s))$$

2022-03-24



Struttura matematica iv		
	TAG	READER
$n_t = rnd()$		$n_t = recv()$
$s = push(n_t \oplus uuid, set(skey, s))$		$s = push(n_t \oplus uuid, set(skey, s))$
$send(n_t)$		$n_r = rnd()$
		$send(n_r \oplus ks(s))$
		$s = push(n_r, s)$
		$send(suc^2(n_t) \oplus ks(s))$

Struttura matematica iv

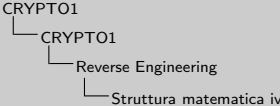
TAG

$$n_t = rnd()$$
$$s = push(n_t \oplus uuid, set(skey, s))$$
$$send(n_t)$$

READER

$$n_t = recv()$$
$$s = push(n_t \oplus uuid, set(skey, s))$$
$$n_r = rnd()$$
$$send(n_r \oplus ks(s))$$
$$s = push(n_r, s)$$
$$send(suc^2(n_t) \oplus ks(s))$$

2022-03-24



Struttura matematica iv		
	TAG	READER
	$n_t = rnd()$	$n_t = recv()$
	$s = push(n_t \oplus uuid, set(skey, s))$	$s = push(n_t \oplus uuid, set(skey, s))$
	$send(n_t)$	$n_r = rnd()$
		$send(n_r \oplus ks(s))$
		$s = push(n_r, s)$
		$send(suc^2(n_t) \oplus ks(s))$

	TAG	READER
	$n_t = rnd()$ $s = push(n_t \oplus uuid, set(skey, s))$ $send(n_t)$	$n_t = recv()$ $s = push(n_t \oplus uuid, set(skey, s))$ $n_r = rnd()$ $send(n_r \oplus ks(s))$ $s = push(n_r, s)$ $send(suc^2(n_t) \oplus ks(s))$

Struttura matematica iv		
	TAG	READER
	$n_t = rnd()$ $s = push(n_t \oplus uuid, set(skey, s))$ $send(n_t)$	$n_t = recv()$ $s = push(n_t \oplus uuid, set(skey, s))$ $n_r = rnd()$ $send(n_r \oplus ks(s))$ $s = push(n_r, s)$ $send(suc^2(n_t) \oplus ks(s))$

Struttura matematica iv

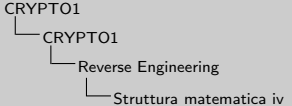
TAG

$$n_t = rnd()$$
$$s = push(n_t \oplus uuid, set(skey, s))$$
$$send(n_t)$$

READER

$$n_t = recv()$$
$$s = push(n_t \oplus uuid, set(skey, s))$$
$$n_r = rnd()$$
$$send(n_r \oplus ks(s))$$
$$s = push(n_r, s)$$
$$send(suc^2(n_t) \oplus ks(s))$$

2022-03-24



Struttura matematica iv		
	TAG	READER
$n_t = rnd()$ $s = push(n_t \oplus uuid, set(skey, s))$ $send(n_t)$		$n_t = recv()$ $s = push(n_t \oplus uuid, set(skey, s))$ $n_r = rnd()$ $send(n_r \oplus ks(s))$ $s = push(n_r, s)$ $send(suc^2(n_t) \oplus ks(s))$

	TAG	READER
1 Sommario		
2 ISO14443		
3 Cifrari di flusso	$n_t = rnd()$ $s = push(n_t \oplus uuid, set(skey, s))$ $send(n_t)$	$n_t = recv()$ $s = push(n_t \oplus uuid, set(skey, s))$ $n_r = rnd()$ $send(n_r \oplus ks(s))$ $s = push(n_r, s)$ $send(suc^2(n_t) \oplus ks(s))$
4 CRYPTO1		
5 Bibliografia		

	TAG	READER
1 Sommario		
2 ISO14443		
3 Cifrari di flusso	$n_t = rnd()$ $s = push(n_t \oplus uuid, set(skey, s))$ $send(n_t)$	$n_t = recv()$ $s = push(n_t \oplus uuid, set(skey, s))$ $n_r = rnd()$ $send(n_r \oplus ks(s))$ $s = push(n_r, s)$ $send(suc^2(n_t) \oplus ks(s))$
4 CRYPTO1		
5 Bibliografia		

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse
Engineering

Filter Function

Attacchi

Considerazioni

Successori

5 Bibliogra-
fia

Struttura matematica v

$$n_r = \textit{recv}() \oplus ks(s)$$

$$s = \textit{push}(n_r, s)$$

$$\textit{assert}(suc^2(n_t) == \textit{recv}() \oplus ks(s))$$

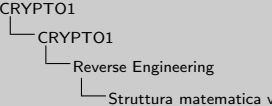
$$s = \textit{push}(n_r, s)$$

$$\textit{send}(suc^3(n_t) \oplus ks(s))$$

$$a_r = \textit{recv}() \oplus ks(s)$$

$$\textit{assert}(suc^3(n_t) == a_r)$$

2022-03-24



Struttura matematica v

```
n_r = recv() ⊕ ks(s)
s = push(n_r, s)
assert(suc2(n_t) == recv() ⊕ ks(s))
s = push(n_r, s)
send(suc3(n_t) ⊕ ks(s))

a_r = recv() ⊕ ks(s)
assert(suc3(n_t) == a_r)
```

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse
Engineering

Filter Function

Attacchi

Considerazioni

Successori

5 Bibliogra-
fia

Struttura matematica v

$$n_r = \textit{recv}() \oplus ks(s)$$

$$s = \textit{push}(n_r, s)$$

$$\textit{assert}(suc^2(n_t) == \textit{recv}() \oplus ks(s))$$

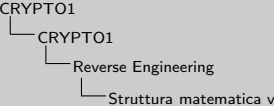
$$s = \textit{push}(n_r, s)$$

$$\textit{send}(suc^3(n_t) \oplus ks(s))$$

$$a_r = \textit{recv}() \oplus ks(s)$$

$$\textit{assert}(suc^3(n_t) == a_r)$$

2022-03-24



Struttura matematica v

```
n_r = recv() ⊕ ks(s)
s = push(n_r, s)
assert(suc2(n_t) == recv() ⊕ ks(s))
s = push(n_r, s)
send(suc3(n_t) ⊕ ks(s))
```

```
a_r = recv() ⊕ ks(s)
assert(suc3(n_t) == a_r)
```

Struttura matematica v

$$n_r = \text{recv}() \oplus ks(s)$$

$$s = \text{push}(n_r, s)$$

$$\text{assert}(\text{suc}^2(n_t) == \text{recv}() \oplus ks(s))$$

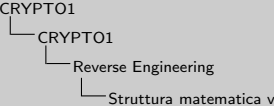
$$s = \text{push}(n_r, s)$$

$$\text{send}(\text{suc}^3(n_t) \oplus ks(s))$$

$$a_r = \text{recv}() \oplus ks(s)$$

$$\text{assert}(\text{suc}^3(n_t) == a_r)$$

2022-03-24



Struttura matematica v

$n_r = \text{recv}() \oplus ks(s)$
 $s = \text{push}(n_r, s)$
 $\text{assert}(\text{suc}^2(n_t) == \text{recv}() \oplus ks(s))$
 $s = \text{push}(n_r, s)$
 $\text{send}(\text{suc}^3(n_t) \oplus ks(s))$

$a_r = \text{recv}() \oplus ks(s)$
 $\text{assert}(\text{suc}^3(n_t) == a_r)$

Struttura matematica v

$$n_r = \text{recv}() \oplus ks(s)$$

$$s = \text{push}(n_r, s)$$

$$\text{assert}(\text{suc}^2(n_t) == \text{recv}() \oplus ks(s))$$

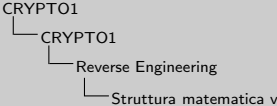
$$s = \text{push}(n_r, s)$$

$$\text{send}(\text{suc}^3(n_t) \oplus ks(s))$$

$$a_r = \text{recv}() \oplus ks(s)$$

$$\text{assert}(\text{suc}^3(n_t) == a_r)$$

2022-03-24



Struttura matematica v

$n_r = \text{recv}() \oplus ks(s)$
 $s = \text{push}(n_r, s)$
 $\text{assert}(\text{suc}^2(n_t) == \text{recv}() \oplus ks(s))$
 $s = \text{push}(n_r, s)$
 $\text{send}(\text{suc}^3(n_t) \oplus ks(s))$

$a_r = \text{recv}() \oplus ks(s)$
 $\text{assert}(\text{suc}^3(n_t) == a_r)$

Struttura matematica v

$$n_r = \text{recv}() \oplus ks(s)$$

$$s = \text{push}(n_r, s)$$

$$\text{assert}(\text{suc}^2(n_t) == \text{recv}() \oplus ks(s))$$

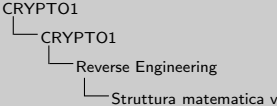
$$s = \text{push}(n_r, s)$$

$$\text{send}(\text{suc}^3(n_t) \oplus ks(s))$$

$$a_r = \text{recv}() \oplus ks(s)$$

$$\text{assert}(\text{suc}^3(n_t) == a_r)$$

2022-03-24



Struttura matematica v

$n_r = \text{recv}() \oplus ks(s)$
 $s = \text{push}(n_r, s)$
 $\text{assert}(\text{suc}^2(n_t) == \text{recv}() \oplus ks(s))$
 $s = \text{push}(n_r, s)$
 $\text{send}(\text{suc}^3(n_t) \oplus ks(s))$

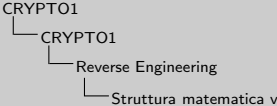
$a_r = \text{recv}() \oplus ks(s)$
 $\text{assert}(\text{suc}^3(n_t) == a_r)$

Struttura matematica v

$$\begin{aligned} n_r &= \text{recv}() \oplus ks(s) \\ s &= \text{push}(n_r, s) \\ \text{assert}(suc^2(n_t) == \text{recv}() \oplus ks(s)) \\ s &= \text{push}(n_r, s) \\ \text{send}(suc^3(n_t) \oplus ks(s)) \end{aligned}$$

$$\begin{aligned} a_r &= \text{recv}() \oplus ks(s) \\ \text{assert}(suc^3(n_t) == a_r) \end{aligned}$$

2022-03-24



Struttura matematica v

```
n_r = recv() ⊕ ks(s)
s = push(n_r, s)
assert(suc²(n_t) == recv() ⊕ ks(s))
s = push(n_r, s)
send(suc³(n_t) ⊕ ks(s))
```

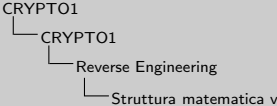
$a_r = \text{recv}() \oplus ks(s)$
 $\text{assert}(suc^3(n_t) == a_r)$

Struttura matematica v

$$\begin{aligned} n_r &= \text{recv}() \oplus ks(s) \\ s &= \text{push}(n_r, s) \\ \text{assert}(suc^2(n_t) == \text{recv}() \oplus ks(s)) \\ s &= \text{push}(n_r, s) \\ \text{send}(suc^3(n_t) \oplus ks(s)) \end{aligned}$$

$$\begin{aligned} a_r &= \text{recv}() \oplus ks(s) \\ \text{assert}(suc^3(n_t) == a_r) \end{aligned}$$

2022-03-24



Struttura matematica v

```
n_r = recv() ⊕ ks(s)
s = push(n_r, s)
assert(suc²(n_t) == recv() ⊕ ks(s))
s = push(n_r, s)
send(suc³(n_t) ⊕ ks(s))

a_r = recv() ⊕ ks(s)
assert(suc³(n_t) == a_r)
```

Analisi della funzione di filtraggio

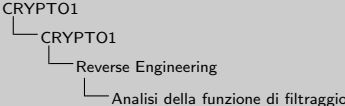
È possibile controllare interamente lo stato del LFSR grazie al fatto che tutti i parametri sono visibili.

Essendo il protocollo di comunicazione documentato, è possibile inviare al lettore/tag dati personalizzati.

È quindi possibile inviare al lettore chiave, UUID e nonce pari sempre a 0 (o a un valore noto) in modo da controllare lo stato α dell’LFSR

Inoltre è possibile mantenere n_r costante riavviando il lettore ogni volta

2022-03-24



È possibile controllare interamente lo stato del LFSR grazie al fatto che tutti i parametri sono visibili.

Essendo il protocollo di comunicazione documentato, è possibile inviare al lettore/tag dati personalizzati.

È quindi possibile inviare al lettore chiave, UUID e nonce pari sempre a 0 (o a un valore noto) in modo da controllare lo stato α dell’LFSR

Inoltre è possibile mantenere n_r costante riavviando il lettore ogni volta

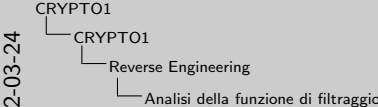
Analisi della funzione di filtraggio

È possibile controllare interamente lo stato del LFSR grazie al fatto che tutti i parametri sono visibili.

Essendo il protocollo di comunicazione documentato, è possibile inviare al lettore/tag dati personalizzati.

È quindi possibile inviare al lettore chiave, UUID e nonce pari sempre a 0 (o a un valore noto) in modo da controllare lo stato α dell’LFSR

Inoltre è possibile mantenere n_r costante riavviando il lettore ogni volta



È possibile controllare interamente lo stato del LFSR grazie al fatto che tutti i parametri sono visibili.

Essendo il protocollo di comunicazione documentato, è possibile inviare al lettore/tag dati personalizzati.

È quindi possibile inviare al lettore chiave, UUID e nonce pari sempre a 0 (o a un valore noto) in modo da controllare lo stato α dell’LFSR

Inoltre è possibile mantenere n_r costante riavviando il lettore ogni volta

Analisi della funzione di filtraggio

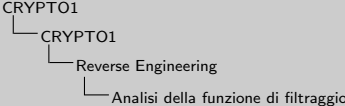
È possibile controllare interamente lo stato del LFSR grazie al fatto che tutti i parametri sono visibili.

Essendo il protocollo di comunicazione documentato, è possibile inviare al lettore/tag dati personalizzati.

È quindi possibile inviare al lettore chiave, UUID e nonce pari sempre a 0 (o a un valore noto) in modo da controllare lo stato α dell’LFSR

Inoltre è possibile mantenere n_r costante riavviando il lettore ogni volta

2022-03-24



È possibile controllare interamente lo stato del LFSR grazie al fatto che tutti i parametri sono visibili.

Essendo il protocollo di comunicazione documentato, è possibile inviare al lettore/tag dati personalizzati.

È quindi possibile inviare al lettore chiave, UUID e nonce pari sempre a 0 (o a un valore noto) in modo da controllare lo stato α dell’LFSR

Inoltre è possibile mantenere n_r costante riavviando il lettore ogni volta

Analisi della funzione di filtraggio

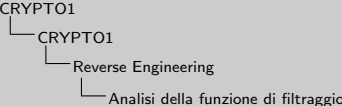
È possibile controllare interamente lo stato del LFSR grazie al fatto che tutti i parametri sono visibili.

Essendo il protocollo di comunicazione documentato, è possibile inviare al lettore/tag dati personalizzati.

È quindi possibile inviare al lettore chiave, UUID e nonce pari sempre a 0 (o a un valore noto) in modo da controllare lo stato α dell’LFSR

Inoltre è possibile mantenere n_r costante riavviando il lettore ogni volta

2022-03-24



È possibile controllare interamente lo stato del LFSR grazie al fatto che tutti i parametri sono visibili.

Essendo il protocollo di comunicazione documentato, è possibile inviare al lettore/tag dati personalizzati.

È quindi possibile inviare al lettore chiave, UUID e nonce pari sempre a 0 (o a un valore noto) in modo da controllare lo stato α dell’LFSR

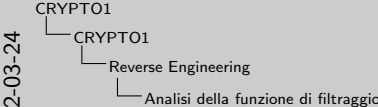
Inoltre è possibile mantenere n_r costante riavviando il lettore ogni volta

Analisi della funzione di filtraggio

È possibile quindi dedurre informazioni sulla funzione f variando di un bit lo stato α .

Se $f(\alpha) \neq f(\alpha')$ allora il bit modificato farà parte degli input della funzione f
Altrimenti non è possibile dedurlo con certezza

Tramite inferenza statistica si può raggiungere una discreta certezza a riguardo



È possibile quindi dedurre informazioni sulla funzione f variando di un bit lo stato α .

Se $f(\alpha) \neq f(\alpha')$ allora il bit modificato farà parte degli input della funzione f

Altrimenti non è possibile dedurlo con certezza

Tramite inferenza statistica si può raggiungere una discreta certezza a riguardo

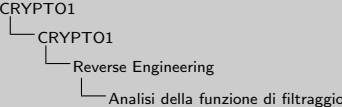
Analisi della funzione di filtraggio

È possibile quindi dedurre informazioni sulla funzione f variando di un bit lo stato α .

Se $f(\alpha) \neq f(\alpha')$ allora il bit modificato farà parte degli input della funzione f
Altrimenti non è possibile dedurlo con certezza

Tramite inferenza statistica si può raggiungere una discreta certezza a riguardo

2022-03-24



È possibile quindi dedurre informazioni sulla funzione f variando di un bit lo stato α .

Se $f(\alpha) \neq f(\alpha')$ allora il bit modificato farà parte degli input della funzione f
Altrimenti non è possibile dedurlo con certezza

*Tramite inferenza statistica si può raggiungere una discreta certezza a riguardo

Analisi della funzione di filtraggio

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse
Engineering

Filter Function

Attacchi

Considerazioni

Successori

5 Bibliogra-
fia

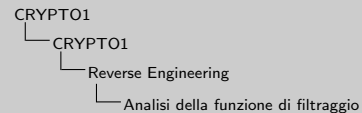
È possibile quindi dedurre informazioni sulla funzione f variando di un bit lo stato α .

Se $f(\alpha) \neq f(\alpha')$ allora il bit modificato farà parte degli input della funzione f

Altrimenti non è possibile dedurlo con certezza

Tramite inferenza statistica si può raggiungere una discreta certezza a riguardo

2022-03-24



Analisi della funzione di filtraggio

È possibile quindi dedurre informazioni sulla funzione f variando di un bit lo stato α .

Se $f(\alpha) \neq f(\alpha')$ allora il bit modificato farà parte degli input della funzione f

Altrimenti non è possibile dedurlo con certezza

Tramite inferenza statistica si può raggiungere una discreta certezza a riguardo

Analisi della funzione di filtraggio

- 1 Sommario
- 2 ISO14443
- 3 Cifrari di flusso
- 4 CRYPTO1
 - Storia
 - Vulnerabilità
 - Reverse Engineering
 - Filter Function
 - Attacchi
 - Considerazioni
 - Successori
- 5 Bibliografia

Dopo una ricerca esaustiva si è giunti a ricavare la seguente funzione f

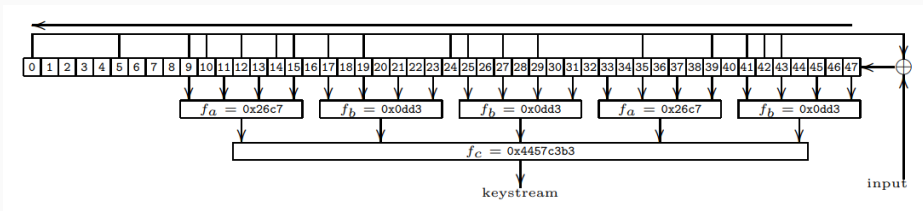
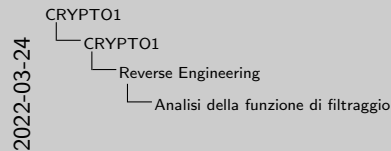


Figura 10: Risultato del reverse engineering e dei test[GKGM⁺08]



Si noti la grande somiglianza con lo stream cipher Hitag2. Questo cifrario è utilizzato solitamente nelle chiavi delle automobili dotate di verifica wireless del dispositivo tramite RFID (125kHz). Anche questo cifrario è stato dichiarato insicuro e vulnerabile.[VGB12]

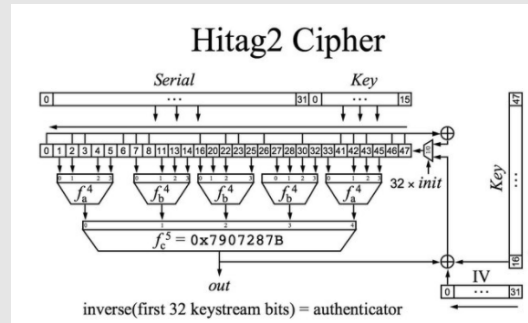


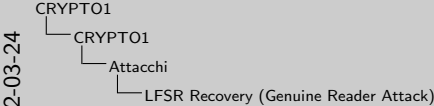
Figura 11: Cifrario Hitag2

LFSR Recovery (Genuine Reader Attack)

Data la dimensione ridotta del LFSR è possibile creare una tabella contenente le coppie ($LFSR_{value}$, $KeyStream$) per tutti gli stati da 0x0000000000000 a 0x000FFFFFFFFF (2³⁶ righe)

Successivamente si effettua un tentativo di autenticazione simulando un tag, inviando all'inizio della trasmissione un nonce n_t nella forma 0x0000XXX0 (4096 tentativi) senza procedere oltre al primo passaggio

Questo causa l'invio di un comando di halt da parte del lettore (comando noto) cifrato con lo stato attuale dal quale possiamo ricavare ks_3 e ks_2 dalla risposta $suc^2(n_t)$



2022-03-24

Data la dimensione ridotta del LFSR è possibile creare una tabella contenente le coppie ($LFSR_{value}$, $KeyStream$) per tutti gli stati da 0x000000000000 a 0x000FFFFFFFFF (2³⁶ righe)

Successivamente si effettua un tentativo di autenticazione simulando un tag, inviando all'inizio della trasmissione un nonce n_t nella forma 0x0000XXX0 (4096 tentativi) senza procedere oltre al primo passaggio

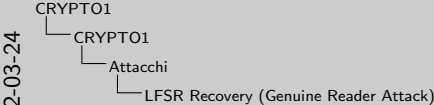
Questo causa l'invio di un comando di halt da parte del lettore (comando noto) cifrato con lo stato attuale dal quale possiamo ricavare ks_3 e ks_2 dalla risposta $suc^2(n_t)$

LFSR Recovery (Genuine Reader Attack)

Data la dimensione ridotta del LFSR è possibile creare una tabella contenente le coppie ($LFSR_{value}$, $KeyStream$) per tutti gli stati da 0x0000000000000 a 0x000FFFFFFFFF (2³⁶ righe)

Successivamente si effettua un tentativo di autenticazione simulando un tag, inviando all’inizio della trasmissione un nonce n_t nella forma 0x0000XXX0 (4096 tentativi) senza procedere oltre al primo passaggio

Questo causa l’invio di un comando di halt da parte del lettore (comando noto) cifrato con lo stato attuale dal quale possiamo ricavare ks_3 e ks_2 dalla risposta $suc^2(n_t)$



Data la dimensione ridotta del LFSR è possibile creare una tabella contenente le coppie ($LFSR_{value}$, $KeyStream$) per tutti gli stati da 0x000000000000 a 0x00FFFFFFFF (2³⁶ righe)

Successivamente si effettua un tentativo di autenticazione simulando un tag, inviando all’inizio della trasmissione un nonce n_t nella forma 0x000XXX0 (4096 tentativi) senza procedere oltre al primo passaggio

Questo causa l’invio di un comando di halt da parte del lettore (comando noto) cifrato con lo stato attuale dal quale possiamo ricavare ks_3 e ks_2 dalla risposta $suc^2(n_t)$

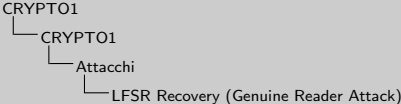
LFSR Recovery (Genuine Reader Attack)

Data la dimensione ridotta del LFSR è possibile creare una tabella contenente le coppie ($LFSR_{value}$, $KeyStream$) per tutti gli stati da 0x000000000000 a 0x000FFFFFFFFF (2^{36} righe)

Successivamente si effettua un tentativo di autenticazione simulando un tag, inviando all’inizio della trasmissione un nonce n_t nella forma 0x0000XXX0 (4096 tentativi) senza procedere oltre al primo passaggio

Questo causa l’invio di un comando di halt da parte del lettore (comando noto) cifrato con lo stato attuale dal quale possiamo ricavare ks_3 e ks_2 dalla risposta $suc^2(n_t)$

2022-03-24



Come è possibile osservare dalla figura 10 la funzione f non dipende dai valori terminali del LFSR.

Data la dimensione ridotta del LFSR è possibile creare una tabella contenente le coppie ($LFSR_{value}$, $KeyStream$) per tutti gli stati da 0x000000000000 a 0x000FFFFFFFFF (2^{36} righe)

Successivamente si effettua un tentativo di autenticazione simulando un tag, inviando all’inizio della trasmissione un nonce n_t nella forma 0x0000XXX0 (4096 tentativi) senza procedere oltre al primo passaggio

Questo causa l’invio di un comando di halt da parte del lettore (comando noto) cifrato con lo stato attuale dal quale possiamo ricavare ks_3 e ks_2 dalla risposta $suc^2(n_t)$

LFSR Recovery (Genuine Reader Attack)

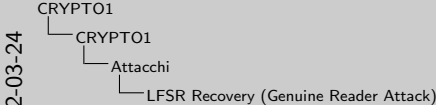
Data la struttura è stato notato come esista un valore di n_t tale per cui lo stato del LFSR equivale a

$$0xYYYYYYYY000Y$$

Il lettore quindi aggiorna il cifrario con il valore di n_r , fa sì che gli zeri nel LFSR raggiungano lo stato

$$0x000YZZZZZZZZ$$

Possiamo quindi trovare lo stato del LFSR dalla tabella cercando per i valori di ks_2 e ks_3



LFSR Recovery (Genuine Reader Attack)

Data la struttura è stato notato come esista un valore di n_t tale per cui lo stato del LFSR equivale a

$$0xYYYYYYYY000Y$$

Il lettore quindi aggiorna il cifrario con il valore di n_r , fa sì che gli zeri nel LFSR raggiungano lo stato

$$0x000YZZZZZZZZ$$

Possiamo quindi trovare lo stato del LFSR dalla tabella cercando per i valori di ks_2 e ks_3

LFSR Recovery (Genuine Reader Attack)

Data la struttura è stato notato come esista un valore di n_t tale per cui lo stato del LFSR equivale a

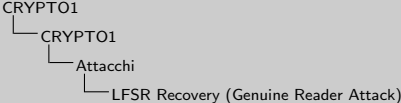
$$0xYYYYYYYY000Y$$

Il lettore quindi aggiorna il cifrario con il valore di n_r , fa sì che gli zeri nel LFSR raggiungano lo stato

$$0x000YZZZZZZZZ$$

Possiamo quindi trovare lo stato del LFSR dalla tabella cercando per i valori di ks_2 e ks_3

2022-03-24



LFSR Recovery (Genuine Reader Attack)

Data la struttura è stato notato come esista un valore di n_t tale per cui lo stato del LFSR equivale a

0xYYYYYYYY000Y

Il lettore quindi aggiorna il cifrario con il valore di n_r , fa sì che gli zeri nel LFSR raggiungano lo stato

0x000YZZZZZZZZ

Possiamo quindi trovare lo stato del LFSR dalla tabella cercando per i valori di ks_2 e ks_3

- 1 Sommario
- 2 ISO14443
- 3 Cifrari di flusso
- 4 CRYPTO1
 - Storia
 - Vulnerabilità
 - Reverse Engineering
 - Attacchi
 - Genuine Reader
 - Genuine Tag
 - Nested Authentication
 - Considerazioni
 - Successori
- 5 Bibliografia

LFSR Recovery (Genuine Reader Attack)

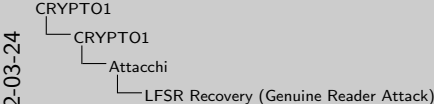
Data la struttura è stato notato come esista un valore di n_t tale per cui lo stato del LFSR equivale a

$$0xYYYYYYYY000Y$$

Il lettore quindi aggiorna il cifrario con il valore di n_r , fa sì che gli zeri nel LFSR raggiungano lo stato

$$0x000YZZZZZZZZ$$

Possiamo quindi trovare lo stato del LFSR dalla tabella cercando per i valori di ks_2 e ks_3



LFSR Recovery (Genuine Reader Attack)	
Data la struttura è stato notato come esista un valore di n_t tale per cui lo stato del LFSR equivale a	0xYYYYYYYY000Y
Il lettore quindi aggiorna il cifrario con il valore di n_r , fa sì che gli zeri nel LFSR raggiungano lo stato	0x000YZZZZZZZZ
Possiamo quindi trovare lo stato del LFSR dalla tabella cercando per i valori di ks_2 e ks_3	

LFSR Rollback (Genuine Reader Attack)

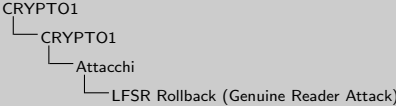
Dato un qualsivoglia stato del LFSR è possibile ottenere lo stato dell’iterazione precedente conoscendo quanto è stato inserito.

Una volta ottenuto lo stato del LFSR dopo l’inserzione del nonce n_r non è ancora possibile effettuare il rollback perchè n_r è cifrato.

La vulnerabilità viene utilizzata ora: dato che l’MSB del LFSR non è input della funzione di filtro f (Vedi figura 10) è possibile sostituirlo con un valore r senza effetti collaterali.

Ciò permette di ottenere il n_{r31} . Utilizzando questo valore è quindi possibile ricalcolare il valore di r

2022-03-24



Dato un qualsivoglia stato del LFSR è possibile ottenere lo stato dell’iterazione precedente conoscendo quanto è stato inserito.

Una volta ottenuto lo stato del LFSR dopo l’inserzione del nonce n_r non è ancora possibile effettuare il rollback perchè n_r è cifrato.

La vulnerabilità viene utilizzata ora: dato che l’MSB del LFSR non è input della funzione di filtro f (Vedi figura 10) è possibile sostituirlo con un valore r senza effetti collaterali.

Ciò permette di ottenere il n_{r31} . Utilizzando questo valore è quindi possibile ricalcolare il valore di r

LFSR Rollback (Genuine Reader Attack)

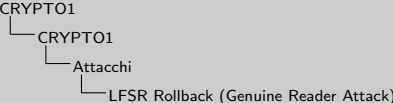
Dato un qualsivoglia stato del LFSR è possibile ottenere lo stato dell’iterazione precedente conoscendo quanto è stato inserito.

Una volta ottenuto lo stato del LFSR dopo l’inserzione del nonce n_r non è ancora possibile effettuare il rollback perchè n_r è cifrato.

La vulnerabilità viene utilizzata ora: dato che l'MSB del LFSR non è input della funzione di filtro f (Vedi figura 10) è possibile sostituirlo con un valore r senza effetti collaterali.

Ciò permette di ottenere il n_{r31} . Utilizzando questo valore è quindi possibile ricalcolare il valore di r

2022-03-24



Dato un qualsivoglia stato del LFSR è possibile ottenere lo stato dell’iterazione precedente conoscendo quanto è stato inserito.
Una volta ottenuto lo stato del LFSR dopo l’inserzione del nonce n_r non è ancora possibile effettuare il rollback perchè n_r è cifrato.
La vulnerabilità viene utilizzata ora: dato che l'MSB del LFSR non è input della funzione di filtro f (Vedi figura 10) è possibile sostituirlo con un valore r senza effetti collaterali.
Ciò permette di ottenere il n_{r31} . Utilizzando questo valore è quindi possibile ricalcolare il valore di r

LFSR Rollback (Genuine Reader Attack)

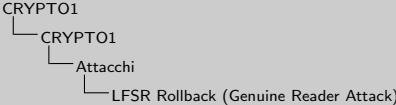
Dato un qualsivoglia stato del LFSR è possibile ottenere lo stato dell’iterazione precedente conoscendo quanto è stato inserito.

Una volta ottenuto lo stato del LFSR dopo l’inserzione del nonce n_r non è ancora possibile effettuare il rollback perchè n_r è cifrato.

La vulnerabilità viene utilizzata ora: dato che l’MSB del LFSR non è input della funzione di filtro f (Vedi figura 10) è possibile sostituirlo con un valore r senza effetti collaterali.

Ciò permette di ottenere il n_{r31} . Utilizzando questo valore è quindi possibile ricalcolare il valore di r

2022-03-24



Dato un qualsivoglia stato del LFSR è possibile ottenere lo stato dell’iterazione precedente conoscendo quanto è stato inserito.
Una volta ottenuto lo stato del LFSR dopo l’inserzione del nonce n_r non è ancora possibile effettuare il rollback perchè n_r è cifrato.
La vulnerabilità viene utilizzata ora: dato che l’MSB del LFSR non è input della funzione di filtro f (Vedi figura 10) è possibile sostituirlo con un valore r senza effetti collaterali.
Ciò permette di ottenere il n_{r31} . Utilizzando questo valore è quindi possibile ricalcolare il valore di r

LFSR Rollback (Genuine Reader Attack)

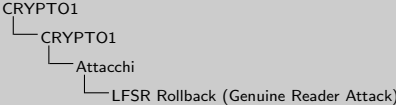
Dato un qualsivoglia stato del LFSR è possibile ottenere lo stato dell’iterazione precedente conoscendo quanto è stato inserito.

Una volta ottenuto lo stato del LFSR dopo l’inserzione del nonce n_r non è ancora possibile effettuare il rollback perchè n_r è cifrato.

La vulnerabilità viene utilizzata ora: dato che l’MSB del LFSR non è input della funzione di filtro f (Vedi figura 10) è possibile sostituirlo con un valore r senza effetti collaterali.

Ciò permette di ottenere il n_{r31} . Utilizzando questo valore è quindi possibile ricalcolare il valore di r

2022-03-24



vedi [GKGM⁺08], la relazione è

$$rk + 48 = rk \oplus rk + 5 \oplus rk + 9 \oplus rk + 10 \oplus rk + 12 \oplus rk + 14 \oplus rk + 15 \oplus$$

$$rk + 17 \oplus rk + 19 \oplus rk + 24 \oplus rk + 27 \oplus rk + 29 \oplus rk + 35 \oplus rk + 39 \oplus rk + 41 \oplus rk + 42 \oplus rk + 43 \oplus i.$$

Dato un qualsivoglia stato del LFSR è possibile ottenere lo stato dell’iterazione precedente conoscendo quanto è stato inserito.
Una volta ottenuto lo stato del LFSR dopo l’inserzione del nonce n_r non è ancora possibile effettuare il rollback perchè n_r è cifrato.
La vulnerabilità viene utilizzata ora: dato che l’MSB del LFSR non è input della funzione di filtro f (Vedi figura 10) è possibile sostituirlo con un valore r senza effetti collaterali.
Ciò permette di ottenere il n_{r31} . Utilizzando questo valore è quindi possibile ricalcolare il valore di r

Genuine Tag Attack

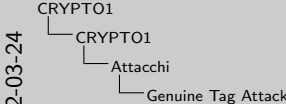
Sfruttando la vulnerabilità relativa al leak di informazioni all’invio di un NACK (slide 24) è possibile ottenere la chiave utilizzata dal tag.

Ottenendo il NACK cifrato è possibile ricavare 4 bit di keystream.

Per fare ciò viene inviata una sequenza di otto byte \bar{c} (la quale dovrebbe contenere n_r e a_r) con i parity bit casuali.

Inviando un massimo di 256 query è possibile trovare la combinazione dei bit di parità tale per cui siano corretti.

È possibile notare che la probabilità che gli ultimi 3 bit del keystream derivato dalla decodifica degli ultimi tre bit di c_3 non dipendano da quest’ultimo; e tale probabilità è di 0.75 [Cou09]



2022-03-24

Genuine Tag Attack

Sfruttando la vulnerabilità relativa al leak di informazioni all’invio di un NACK (slide 24) è possibile ottenere la chiave utilizzata dal tag.

Ottenendo il NACK cifrato è possibile ricavare 4 bit di keystream.

Per fare ciò viene inviata una sequenza di otto byte \bar{c} (la quale dovrebbe contenere n_r e a_r) con i parity bit casuali.

Inviando un massimo di 256 query è possibile trovare la combinazione dei bit di parità tale per cui siano corretti.

È possibile notare che la probabilità che gli ultimi 3 bit del keystream derivato dalla decodifica degli ultimi tre bit di c_3 non dipendano da quest’ultimo; e tale probabilità è di 0.75 [Cou09]

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse
Engineering

Attacchi

Genuine Reader

Genuine Tag

Nested
Authentication

Considerazioni

Successori

5 Bibliogra-
fia

Genuine Tag Attack

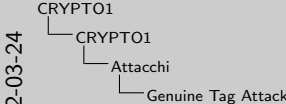
Sfruttando la vulnerabilità relativa al leak di informazioni all’invio di un NACK (slide 24) è possibile ottenere la chiave utilizzata dal tag.

Ottenendo il NACK cifrato è possibile ricavare 4 bit di keystream.

Per fare ciò viene inviata una sequenza di otto byte \bar{c} (la quale dovrebbe contenere n_r e a_r) con i parity bit casuali.

Inviando un massimo di 256 query è possibile trovare la combinazione dei bit di parità tale per cui siano corretti.

È possibile notare che la probabilità che gli ultimi 3 bit del keystream derivato dalla decodifica degli ultimi tre bit di c_3 non dipendano da quest’ultimo; e tale probabilità è di 0.75 [Cou09]



2022-03-24

Genuine Tag Attack

Sfruttando la vulnerabilità relativa al leak di informazioni all’invio di un NACK (slide 24) è possibile ottenere la chiave utilizzata dal tag.

Ottenendo il NACK cifrato è possibile ricavare 4 bit di keystream.

Per fare ciò viene inviata una sequenza di otto byte \bar{c} (la quale dovrebbe contenere n_r e a_r) con i parity bit casuali.

Inviando un massimo di 256 query è possibile trovare la combinazione dei bit di parità tale per cui siano corretti.

È possibile notare che la probabilità che gli ultimi 3 bit del keystream derivato dalla decodifica degli ultimi tre bit di c_3 non dipendano da quest’ultimo; e tale probabilità è di 0.75 [Cou09]

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse
Engineering

Attacchi

Genuine Reader

Genuine Tag

Nested
Authentication

Considerazioni

Successori

5 Bibliogra-
fia

Genuine Tag Attack

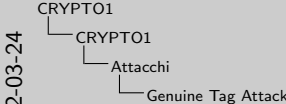
Sfruttando la vulnerabilità relativa al leak di informazioni all’invio di un NACK (slide 24) è possibile ottenere la chiave utilizzata dal tag.

Ottenendo il NACK cifrato è possibile ricavare 4 bit di keystream.

Per fare ciò viene inviata una sequenza di otto byte \bar{c} (la quale dovrebbe contenere n_r e a_r) con i parity bit casuali.

Inviando un massimo di 256 query è possibile trovare la combinazione dei bit di parità tale per cui siano corretti.

È possibile notare che la probabilità che gli ultimi 3 bit del keystream derivato dalla decodifica degli ultimi tre bit di c_3 non dipendano da quest’ultimo; e tale probabilità è di 0.75 [Cou09]



Genuine Tag Attack

Sfruttando la vulnerabilità relativa al leak di informazioni all’invio di un NACK (slide 24) è possibile ottenere la chiave utilizzata dal tag.

Ottenendo il NACK cifrato è possibile ricavare 4 bit di keystream.

Per fare ciò viene inviata una sequenza di otto byte \bar{c} (la quale dovrebbe contenere n_r e a_r) con i parity bit casuali.

Inviando un massimo di 256 query è possibile trovare la combinazione dei bit di parità tale per cui siano corretti.

È possibile notare che la probabilità che gli ultimi 3 bit del keystream derivato dalla decodifica degli ultimi tre bit di c_3 non dipendano da quest’ultimo; e tale probabilità è di 0.75 [Cou09]

1	Sommario
2	ISO14443
3	Cifrari di flusso
4	CRYPTO1
	Storia
	Vulnerabilità
	Reverse Engineering
	Attacchi
	Genuine Reader
	Genuine Tag
	Nested Authentication
	Considerazioni
	Successori
5	Bibliografia

Genuine Tag Attack

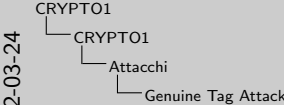
Sfruttando la vulnerabilità relativa al leak di informazioni all’invio di un NACK (slide 24) è possibile ottenere la chiave utilizzata dal tag.

Ottenendo il NACK cifrato è possibile ricavare 4 bit di keystream.

Per fare ciò viene inviata una sequenza di otto byte \bar{c} (la quale dovrebbe contenere n_r e a_r) con i parity bit casuali.

Inviando un massimo di 256 query è possibile trovare la combinazione dei bit di parità tale per cui siano corretti.

È possibile notare che la probabilità che gli ultimi 3 bit del keystream derivato dalla decodifica degli ultimi tre bit di c_3 non dipendano da quest’ultimo; e tale probabilità è di 0.75 [Cou09]



Genuine Tag Attack

Sfruttando la vulnerabilità relativa al leak di informazioni all’invio di un NACK (slide 24) è possibile ottenere la chiave utilizzata dal tag.

Ottenendo il NACK cifrato è possibile ricavare 4 bit di keystream.

Per fare ciò viene inviata una sequenza di otto byte \bar{c} (la quale dovrebbe contenere n_r e a_r) con i parity bit casuali.

Inviando un massimo di 256 query è possibile trovare la combinazione dei bit di parità tale per cui siano corretti.

È possibile notare che la probabilità che gli ultimi 3 bit del keystream derivato dalla decodifica degli ultimi tre bit di c_3 non dipendano da quest’ultimo; e tale probabilità è di 0.75 [Cou09]

Genuine Tag Attack

Sfruttando la vulnerabilità relativa al leak di informazioni all’invio di un NACK (slide 24) è possibile ottenere la chiave utilizzata dal tag.

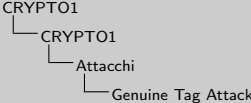
Ottenendo il NACK cifrato è possibile ricavare 4 bit di keystream.

Per fare ciò viene inviata una sequenza di otto byte \bar{c} (la quale dovrebbe contenere n_r e a_r) con i parity bit casuali.

Inviando un massimo di 256 query è possibile trovare la combinazione dei bit di parità tale per cui siano corretti.

È possibile notare che la probabilità che gli ultimi 3 bit del keystream derivato dalla decodifica degli ultimi tre bit di c_3 non dipendano da quest’ultimo; e tale probabilità è di 0.75 [Cou09]

2022-03-24



Genuine Tag Attack

Sfruttando la vulnerabilità relativa al leak di informazioni all’invio di un NACK (slide 24) è possibile ottenere la chiave utilizzata dal tag.

Ottenendo il NACK cifrato è possibile ricavare 4 bit di keystream.

Per fare ciò viene inviata una sequenza di otto byte \bar{c} (la quale dovrebbe contenere n_r e a_r) con i parity bit casuali.

Inviando un massimo di 256 query è possibile trovare la combinazione dei bit di parità tale per cui siano corretti.

È possibile notare che la probabilità che gli ultimi 3 bit del keystream derivato dalla decodifica degli ultimi tre bit di c_3 non dipendano da quest’ultimo; e tale probabilità è di 0.75 [Cou09]

Genuine Tag Attack

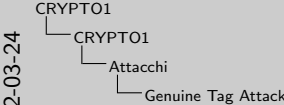
Ogni bit futuro dello stato del LFSR è dato da una combinazione lineare nota di alcuni bit dello stato attuale.

Parte di questi è $UID \oplus n_t$ e sono noti. L'altra parte è n_r il quale è sconosciuto.

Per fare ciò è possibile sfruttare la vulnerabilità dell’RNG per avere sempre lo stesso nonce n_t .

Mantenendo n_t costante insieme ai primi 29 bit della risposta ritentiamo l'autenticazione per le 8 volte (variando quindi tre bit della risposta), variando anche i bit di parità. Dopo aver provato i 2^5 casi otterremo una risposta 1/32 volte.

Ripetendo questo processo possiamo così ottenere 8 risposte con i successivi 8 valori di n_r .



Ogni bit futuro dello stato del LFSR è dato da una combinazione lineare nota di alcuni bit dello stato attuale.

Parte di questi è $UID \oplus n_t$ e sono noti. L'altra parte è n_r il quale è sconosciuto.

Per fare ciò è possibile sfruttare la vulnerabilità dell’RNG per avere sempre lo stesso nonce n_t .

Mantenendo n_t costante insieme ai primi 29 bit della risposta ritentiamo l'autenticazione per le 8 volte (variando quindi tre bit della risposta), variando anche i bit di parità. Dopo aver provato i 2^5 casi otterremo una risposta 1/32 volte.

Ripetendo questo processo possiamo così ottenere 8 risposte con i successivi 8 valori di n_r .

Genuine Tag Attack

Ogni bit futuro dello stato del LFSR è dato da una combinazione lineare nota di alcuni bit dello stato attuale.

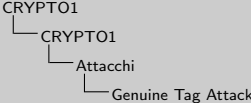
Parte di questi è $UID \oplus n_t$ e sono noti. L'altra parte è n_r il quale è sconosciuto.

Per fare ciò è possibile sfruttare la vulnerabilità dell’RNG per avere sempre lo stesso nonce n_t .

Mantenendo n_t costante insieme ai primi 29 bit della risposta ritentiamo l'autenticazione per le 8 volte (variando quindi tre bit della risposta), variando anche i bit di parità. Dopo aver provato i 2^5 casi otterremo una risposta 1/32 volte.

Ripetendo questo processo possiamo così ottenere 8 risposte con i successivi 8 valori di n_r .

2022-03-24



Ogni bit futuro dello stato del LFSR è dato da una combinazione lineare nota di alcuni bit dello stato attuale.

Parte di questi è $UID \oplus n_t$ e sono noti. L'altra parte è n_r il quale è sconosciuto.

Per fare ciò è possibile sfruttare la vulnerabilità dell’RNG per avere sempre lo stesso nonce n_t .

Mantenendo n_t costante insieme ai primi 29 bit della risposta ritentiamo l'autenticazione per le 8 volte (variando quindi tre bit della risposta), variando anche i bit di parità. Dopo aver provato i 2^5 casi otterremo una risposta 1/32 volte.

Ripetendo questo processo possiamo così ottenere 8 risposte con i successivi 8 valori di n_r .

Genuine Tag Attack

Ogni bit futuro dello stato del LFSR è dato da una combinazione lineare nota di alcuni bit dello stato attuale.

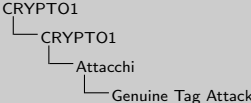
Parte di questi è $UID \oplus n_t$ e sono noti. L'altra parte è n_r il quale è sconosciuto.

Per fare ciò è possibile sfruttare la vulnerabilità dell’RNG per avere sempre lo stesso nonce n_t .

Mantenendo n_t costante insieme ai primi 29 bit della risposta ritentiamo l'autenticazione per le 8 volte (variando quindi tre bit della risposta), variando anche i bit di parità. Dopo aver provato i 2^5 casi otterremo una risposta 1/32 volte.

Ripetendo questo processo possiamo così ottenere 8 risposte con i successivi 8 valori di n_r .

2022-03-24



Ogni bit futuro dello stato del LFSR è dato da una combinazione lineare nota di alcuni bit dello stato attuale.
Parte di questi è $UID \oplus n_t$ e sono noti. L'altra parte è n_r il quale è sconosciuto.
Per fare ciò è possibile sfruttare la vulnerabilità dell’RNG per avere sempre lo stesso nonce n_t .
Mantenendo n_t costante insieme ai primi 29 bit della risposta ritentiamo l'autenticazione per le 8 volte (variando quindi tre bit della risposta), variando anche i bit di parità. Dopo aver provato i 2^5 casi otterremo una risposta 1/32 volte.
Ripetendo questo processo possiamo così ottenere 8 risposte con i successivi 8 valori di n_r .

Genuine Tag Attack

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

- Storia
- Vulnerabilità
- Reverse Engineering
- Attacchi
 - Genuine Reader
 - Genuine Tag**
 - Nested Authentication
- Considerazioni
- Successori

5 Bibliogra-
fia

È ora possibile calcolare la differenza negli stati nelle 8 risposte, confrontandola con valori precomputati. Otteniamo così i bit dispari dello stato

Ripetendo l'operazione possiamo recuperare i bit pari ed effettuare un RollBack del LFSR per ottenere la chiave

2022-03-24

CRYPTO1

CRYPTO1

Attacchi

Genuine Tag Attack

Genuine Tag Attack

È ora possibile calcolare la differenza negli stati nelle 8 risposte, confrontandola con valori precomputati. Otteniamo così i bit dispari dello stato

Ripetendo l'operazione possiamo recuperare i bit pari ed effettuare un RollBack del LFSR per ottenere la chiave

Genuine Tag Attack

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

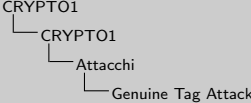
- Storia
- Vulnerabilità
- Reverse Engineering
- Attacchi
 - Genuine Reader
 - Genuine Tag**
 - Nested Authentication
- Considerazioni
- Successori

5 Bibliogra-
fia

È ora possibile calcolare la differenza negli stati nelle 8 risposte, confrontandola con valori precomputati. Otteniamo così i bit dispari dello stato

Ripetendo l'operazione possiamo recuperare i bit pari ed effettuare un RollBack del LFSR per ottenere la chiave

2022-03-24



Genuine Tag Attack

È ora possibile calcolare la differenza negli stati nelle 8 risposte, confrontandola con valori precomputati. Otteniamo così i bit dispari dello stato
Ripetendo l'operazione possiamo recuperare i bit pari ed effettuare un RollBack del LFSR per ottenere la chiave

Genuine Tag Attack

1 Sommario

2 ISO14443

3 Cifrari di
flusso

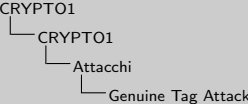
4 CRYPTO1

- Storia
- Vulnerabilità
- Reverse Engineering
- Attacchi
 - Genuine Reader
 - Genuine Tag**
 - Nested Authentication
- Considerazioni
- Successori

5 Bibliogra-
fia

È possibile trovare un’implementazione qui:
<https://github.com/DrSchottky/mfcuk/>

2022-03-24



Genuine Tag Attack

È possibile trovare un'implementazione qui:
<https://github.com/DrSchottky/mfcuk/>

Nested authentication

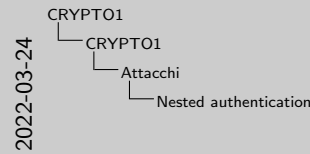
Autenticazioni successive seguono lo stesso protocollo ma n_t viene inviato cifrato con il nuovo stato derivato dalla chiave di autenticazione caricata nel LFSR

Un possibile attacco potrebbe essere il bruteforcing dei 65536 possibili valori casuali

Utilizzando la vulnerabilità dove i bit di parità sono cifrati con un bit condiviso del keystream (Slide 23) possiamo dedurre che

$$n_t = b_{31}, b_{30}, \dots b_1, b_0$$

$$transmission_s tream = \{ b_0, b_1 \dots b_7, p_0, b_8 \dots b_{15}, p_1, \\ b_{16} \dots b_{23}, p_2, b_{24} \dots b_{31}, p_3 \}$$



Nested authentication

Autenticazioni successive seguono lo stesso protocollo ma n_t viene inviato cifrato con il nuovo stato derivato dalla chiave di autenticazione caricata nel LFSR

Un possibile attacco potrebbe essere il bruteforcing dei 65536 possibili valori casuali

Utilizzando la vulnerabilità dove i bit di parità sono cifrati con un bit condiviso del keystream (Slide 23) possiamo dedurre che

$$n_t = b_{31}, b_{30}, \dots b_1, b_0$$

$$transmission_s tream = \{ b_0, b_1 \dots b_7, p_0, b_8 \dots b_{15}, p_1, \\ b_{16} \dots b_{23}, p_2, b_{24} \dots b_{31}, p_3 \}$$

Nested authentication

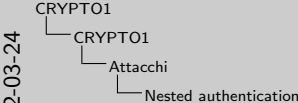
Autenticazioni successive seguono lo stesso protocollo ma n_t viene inviato cifrato con il nuovo stato derivato dalla chiave di autenticazione caricata nel LFSR

Un possibile attacco potrebbe essere il bruteforcing dei 65536 possibili valori casuali

Utilizzando la vulnerabilità dove i bit di parità sono cifrati con un bit condiviso del keystream (Slide 23) possiamo dedurre che

$$n_t = b_{31}, b_{30}, \dots b_1, b_0$$

$$transmission_s tream = \{ b_0, b_1 \dots b_7, p_0, b_8 \dots b_{15}, p_1, \\ b_{16} \dots b_{23}, p_2, b_{24} \dots b_{31}, p_3 \}$$



Nested authentication

Autenticazioni successive seguono lo stesso protocollo ma n_t viene inviato cifrato con il nuovo stato derivato dalla chiave di autenticazione caricata nel LFSR

Un possibile attacco potrebbe essere il bruteforcing dei 65536 possibili valori casuali

Utilizzando la vulnerabilità dove i bit di parità sono cifrati con un bit condiviso del keystream (Slide 23) possiamo dedurre che

$$n_t = b_{31}, b_{30}, \dots b_1, b_0$$

$$transmission_s tream = \{ b_0, b_1 \dots b_7, p_0, b_8 \dots b_{15}, p_1, \\ b_{16} \dots b_{23}, p_2, b_{24} \dots b_{31}, p_3 \}$$

Nested authentication

Autenticazioni successive seguono lo stesso protocollo ma n_t viene inviato cifrato con il nuovo stato derivato dalla chiave di autenticazione caricata nel LFSR

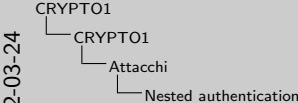
Un possibile attacco potrebbe essere il bruteforcing dei 65536 possibili valori casuali

Utilizzando la vulnerabilità dove i bit di parità sono cifrati con un bit condiviso del keystream (Slide 23) possiamo dedurre che

$$n_t = b_{31}, b_{30}, \dots b_1, b_0$$

$$transmission_s tream = \{ b_0, b_1 \dots b_7, p_0, b_8 \dots b_{15}, p_1,$$

$$b_{16} \dots b_{23}, p_2, b_{24} \dots b_{31}, p_3 \}$$



Nested authentication

Autenticazioni successive seguono lo stesso protocollo ma n_t viene inviato cifrato con il nuovo stato derivato dalla chiave di autenticazione caricata nel LFSR
Un possibile attacco potrebbe essere il bruteforcing dei 65536 possibili valori casuali
Utilizzando la vulnerabilità dove i bit di parità sono cifrati con un bit condiviso del keystream (Slide 23) possiamo dedurre che

$$n_t = b_{31}, b_{30}, \dots b_1, b_0$$

$$transmission_s tream = \{ b_0, b_1 \dots b_7, p_0, b_8 \dots b_{15}, p_1,$$

$$b_{16} \dots b_{23}, p_2, b_{24} \dots b_{31}, p_3 \}$$

Nested authentication

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

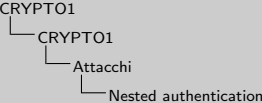
- Storia
- Vulnerabilità
- Reverse Engineering
- Attacchi
 - Genuine Reader
 - Genuine Tag
 - Nested Authentication**
- Considerazioni
- Successori

5 Bibliografia

Dalla precedente vale

$$\left\{ \begin{array}{l} p_0 = rp_0 \oplus ks_8 \\ b_8 = rb_8 \oplus ks_8 \\ p_1 = rp_1 \oplus ks_{16} \\ b_{16} = rb_{16} \oplus ks_{16} \\ p_2 = rp_2 \oplus ks_{24} \\ b_{24} = rb_{24} \oplus ks_{24} \end{array} \right.$$

2022-03-24



Nested authentication

$$\left\{ \begin{array}{l} p_0 = rp_0 \oplus ks_8 \\ b_8 = rb_8 \oplus ks_8 \\ p_1 = rp_1 \oplus ks_{16} \\ b_{16} = rb_{16} \oplus ks_{16} \\ p_2 = rp_2 \oplus ks_{24} \\ b_{24} = rb_{24} \oplus ks_{24} \end{array} \right.$$

Nested authentication

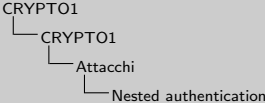
Dalla quale possiamo dedurre

$$\begin{cases} p_0 \oplus b_8 = rb_8 \oplus rp_0 \\ p_1 \oplus b_{16} = rb_{16} \oplus rp_1 \\ p_2 \oplus b_{24} = rb_{24} \oplus rp_2 \end{cases}$$

Di conseguenza è possibile ricavare se il bit di parità del byte precedente è uguale al primo bit del secondo byte, dimezzando lo spazio di ricerca per ogni bit di parità (per un totale di una riduzione di un fattore pari a 8)

Nel caso di una comunicazione tra lettore e tag genuini è possibile ridurre lo spazio di ricerca ulteriormente grazie alla successiva risposta del tag contenente 7 bit di parità, portando così i nonce candidati a 64.

2022-03-24



Nested authentication

Dalla quale possiamo dedurre

$$\begin{cases} p_0 \oplus b_8 = rb_8 \oplus rp_0 \\ p_1 \oplus b_{16} = rb_{16} \oplus rp_1 \\ p_2 \oplus b_{24} = rb_{24} \oplus rp_2 \end{cases}$$

Di conseguenza è possibile ricavare se il bit di parità del byte precedente è uguale al primo bit del secondo byte, dimezzando lo spazio di ricerca per ogni bit di parità (per un totale di una riduzione di un fattore pari a 8)

Nel caso di una comunicazione tra lettore e tag genuini è possibile ridurre lo spazio di ricerca ulteriormente grazie alla successiva risposta del tag contenente 7 bit di parità, portando così i nonce candidati a 64.

Nested authentication

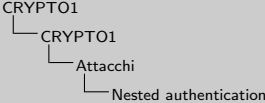
Dalla quale possiamo dedurre

$$\begin{cases} p_0 \oplus b_8 = rb_8 \oplus rp_0 \\ p_1 \oplus b_{16} = rb_{16} \oplus rp_1 \\ p_2 \oplus b_{24} = rb_{24} \oplus rp_2 \end{cases}$$

Di conseguenza è possibile ricavare se il bit di parità del byte precedente è uguale al primo bit del secondo byte, dimezzando lo spazio di ricerca per ogni bit di parità (per un totale di una riduzione di un fattore pari a 8)

Nel caso di una comunicazione tra lettore e tag genuini è possibile ridurre lo spazio di ricerca ulteriormente grazie alla successiva risposta del tag contenente 7 bit di parità, portando così i nonce candidati a 64.

2022-03-24



Nested authentication

Dalla quale possiamo dedurre

$$\begin{cases} p_0 \oplus b_8 = rb_8 \oplus rp_0 \\ p_1 \oplus b_{16} = rb_{16} \oplus rp_1 \\ p_2 \oplus b_{24} = rb_{24} \oplus rp_2 \end{cases}$$

Di conseguenza è possibile ricavare se il bit di parità del byte precedente è uguale al primo bit del secondo byte, dimezzando lo spazio di ricerca per ogni bit di parità (per un totale di una riduzione di un fattore pari a 8)

Nel caso di una comunicazione tra lettore e tag genuini è possibile ridurre lo spazio di ricerca ulteriormente grazie alla successiva risposta del tag contenente 7 bit di parità, portando così i nonce candidati a 64.

Nested authentication

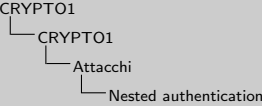
Dalla quale possiamo dedurre

$$\begin{cases} p_0 \oplus b_8 = rb_8 \oplus rp_0 \\ p_1 \oplus b_{16} = rb_{16} \oplus rp_1 \\ p_2 \oplus b_{24} = rb_{24} \oplus rp_2 \end{cases}$$

Di conseguenza è possibile ricavare se il bit di parità del byte precedente è uguale al primo bit del secondo byte, dimezzando lo spazio di ricerca per ogni bit di parità (per un totale di una riduzione di un fattore pari a 8)

Nel caso di una comunicazione tra lettore e tag genuini è possibile ridurre lo spazio di ricerca ulteriormente grazie alla successiva risposta del tag contenente 7 bit di parità, portando così i nonce candidati a 64.

2022-03-24



Adizionalmente è possibile prevedere il valore del nonce data la bassa entropia del rng e la sua predicibilità

Nested authentication

Dalla quale possiamo dedurre

$$\begin{cases} p_0 \oplus b_8 = rb_8 \oplus rp_0 \\ p_1 \oplus b_{16} = rb_{16} \oplus rp_1 \\ p_2 \oplus b_{24} = rb_{24} \oplus rp_2 \end{cases}$$

Di conseguenza è possibile ricavare se il bit di parità del byte precedente è uguale al primo bit del secondo byte, dimezzando lo spazio di ricerca per ogni bit di parità (per un totale di una riduzione di un fattore pari a 8)

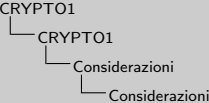
Nel caso di una comunicazione tra lettore e tag genuini è possibile ridurre lo spazio di ricerca ulteriormente grazie alla successiva risposta del tag contenente 7 bit di parità, portando così i nonce candidati a 64.

Considerazioni

MIFARE Classic non rappresenta una piattaforma sicura per lo sviluppo di applicazioni contactless, e in particolare è inadatto ad applicazioni relative a micropagamenti

Durante la progettazione della tecnologia sono stati commessi gravi errori che, data la banalità di alcuni, potrebbero essere stati inseriti con scopi malevoli [Cou09]

2022-03-24



Considerazioni

MIFARE Classic non rappresenta una piattaforma sicura per lo sviluppo di applicazioni contactless, e in particolare è inadatto ad applicazioni relative a micropagamenti

Durante la progettazione della tecnologia sono stati commessi gravi errori che, data la banalità di alcuni, potrebbero essere stati inseriti con scopi malevoli [Cou09]

Considerazioni	
1	Sommario
2	ISO14443
3	Cifrari di flusso
4	CRYPTO1
	Storia
	Vulnerabilità
	Reverse Engineering
	Attacchi
	Considerazioni
	RNG
	TRNG
	LFSR
	Grain128
	Successori
5	Bibliografia

2022-03-24	CRYPTO1
	CRYPTO1
	Considerazioni
	Considerazioni
Considerazioni	
MIFARE Classic non rappresenta una piattaforma sicura per lo sviluppo di applicazioni contactless, e in particolare è inadatto ad applicazioni relative a micropagamenti	
Durante la progettazione della tecnologia sono stati commessi gravi errori che, data la banalità di alcuni, potrebbero essere stati inseriti con scopi malevoli [Cou09]	

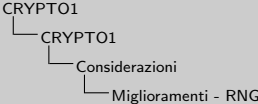
Miglioramenti - RNG

La maggioranza degli attacchi utilizza la predicibilità del RNG.

La soluzione in questo caso è di utilizzare un TRNG disponibile in hardware, a costo di spazio su silicio e di costi pecuniari maggiorati.

Un miglioramento parziale potrebbe avvenire utilizzando un LFSR da 32 bit e non 16, aumentando così le possibili combinazioni al fine di rallentare gli attacchi.

2022-03-24



La maggioranza degli attacchi utilizza la predicibilità del RNG.

La soluzione in questo caso è di utilizzare un TRNG disponibile in hardware, a costo di spazio su silicio e di costi pecuniari maggiorati.

Un miglioramento parziale potrebbe avvenire utilizzando un LFSR da 32 bit e non 16, aumentando così le possibili combinazioni al fine di rallentare gli attacchi.

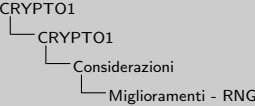
Miglioramenti - RNG

La maggioranza degli attacchi utilizza la predicibilità del RNG.

La soluzione in questo caso è di utilizzare un TRNG disponibile in hardware, a costo di spazio su silicio e di costi pecuniari maggiorati.

Un miglioramento parziale potrebbe avvenire utilizzando un LFSR da 32 bit e non 16, aumentando così le possibili combinazioni al fine di rallentare gli attacchi.

2022-03-24



La maggioranza degli attacchi utilizza la predicibilità del RNG.
La soluzione in questo caso è di utilizzare un TRNG disponibile in hardware, a costo di spazio su silicio e di costi pecuniari maggiorati.
Un miglioramento parziale potrebbe avvenire utilizzando un LFSR da 32 bit e non 16, aumentando così le possibili combinazioni al fine di rallentare gli attacchi.

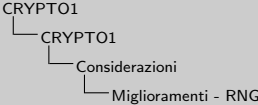
Miglioramenti - RNG

La maggioranza degli attacchi utilizza la predicibilità del RNG.

La soluzione in questo caso è di utilizzare un TRNG disponibile in hardware, a costo di spazio su silicio e di costi pecuniari maggiorati.

Un miglioramento parziale potrebbe avvenire utilizzando un LFSR da 32 bit e non 16, aumentando così le possibili combinazioni al fine di rallentare gli attacchi.

2022-03-24



La maggioranza degli attacchi utilizza la predicibilità del RNG.
La soluzione in questo caso è di utilizzare un TRNG disponibile in hardware, a costo di spazio su silicio e di costi pecuniari maggiorati.
Un miglioramento parziale potrebbe avvenire utilizzando un LFSR da 32 bit e non 16, aumentando così le possibili combinazioni al fine di rallentare gli attacchi.

TRNG i

- 1 Sommario
- 2 ISO14443
- 3 Cifrari di flusso
- 4 CRYPTO1
 - Storia
 - Vulnerabilità
 - Reverse Engineering
 - Attacchi
 - Considerazioni
 - RNG
 - TRNG**
 - LFSR
 - Grain128
 - Successori
- 5 Bibliografia

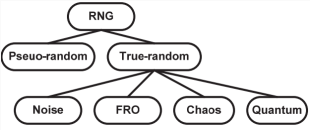
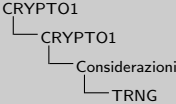


Figura 12: Famiglie di RNG

È possibile creare TRNG basati sul rumore dell’ambiente in relativamente poco spazio e a basso costo. [FA16]

Il concetto principale nella generazione consiste nel generare un rumore bianco casuale dall'ambiente circostante o da una giunzione PN.

2022-03-24



TRNG i



Figura 12: Famiglie di RNG

È possibile creare TRNG basati sul rumore dell’ambiente in relativamente poco spazio e a basso costo. [FA16]
Il concetto principale nella generazione consiste nel generare un rumore bianco casuale dall’ambiente circostante o da una giunzione PN.

TRNG ii

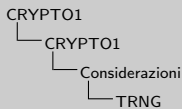
Ciò permette di generare valori metastabili all’ingresso di flipflop che potranno generare risultati non deterministici per gli effetti elettronici interni.



Figura 13: Processo di generazione di un numero casuale.

Da quanto riportato da [FA16] è possibile implementare un TRNG con componenti comuni e a bassissimo costo (LM393) e un microcontrollore (già presente nel tag).

2022-03-24



TRNG ii

Ciò permette di generare valori metastabili all’ingresso di flipflop che potranno generare risultati non deterministici per gli effetti elettronici interni.



Figura 13: Processo di generazione di un numero casuale.

Da quanto riportato da [FA16] è possibile implementare un TRNG con componenti comuni e a bassissimo costo (LM393) e un microcontrollore (già presente nel tag).

TRNG iii

1 Sommario

2 ISO14443

3 Cifrari di flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse Engineering

Attacchi

Considerazioni

RNG

TRNG

LFSR

Grain128

Successori

5 Bibliografia

Senza andare nel dettaglio del funzionamento, il primo comparatore viene utilizzato come vera e propria fonte di entropia, mentre il secondo viene utilizzato come digitalizzatore per avere un output binario.

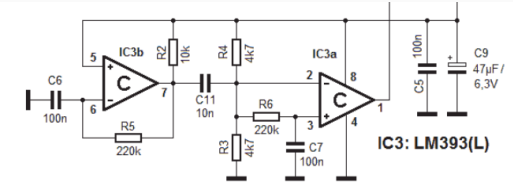
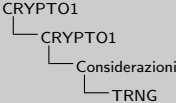


Figura 14: Schema elettronico del generatore di rumore.[FA16]

2022-03-24



TRNG iii

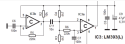


Figura 14: Schema elettronico del generatore di rumore.[FA16]

Senza andare nel dettaglio del funzionamento, il primo comparatore viene utilizzato come vera e propria fonte di entropia, mentre il secondo viene utilizzato come digitalizzatore per avere un output binario.

1	Sommario
2	ISO14443
3	Cifrari di flusso
4	CRYPTO1
	Storia
	Vulnerabilità
	Reverse Engineering
	Attacchi
	Considerazioni
	RNG
	TRNG
	LFSR
	Grain128
	Successori
5	Bibliografia

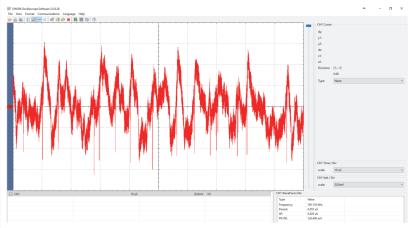
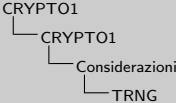


Figura 15: Traccia del rumore generato.[FA16]

Svantaggi: Il consumo di un generico TRNG si aggira sui 100mW [FA16]. Sono necessarie modifiche e ottimizzazioni per l’inclusione in un tag passivo.

2022-03-24



TRNG iv

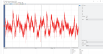


Figura 15: Traccia del rumore generato [FA16]

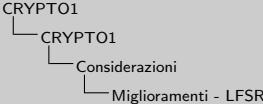
Svantaggi: Il consumo di un generico TRNG si aggira sui 100mW [FA16]. Sono necessarie modifiche e ottimizzazioni per l’inclusione in un tag passivo.

Miglioramenti - LFSR

Alcune vulnerabilità sono causate dalla funzione di filtraggio del LFSR (Slide 32). A tal fine potrebbe essere vantaggiosa un’implementazione dove il cifrario venga sostituito da un modello crittograficamente sicuro.

Una valida proposta potrebbe essere GRAIN-128, cifrario ideato sostanzialmente per ambienti e dispositivi a basso costo e bassissima area occupata[gHJM11][SHSK19]

2022-03-24



Alcune vulnerabilità sono causate dalla funzione di filtraggio del LFSR (Slide 32). A tal fine potrebbe essere vantaggiosa un’implementazione dove il cifrario venga sostituito da un modello crittograficamente sicuro.

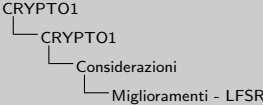
Una valida proposta potrebbe essere GRAIN-128, cifrario ideato sostanzialmente per ambienti e dispositivi a basso costo e bassissima area occupata[gHJM11][SHSK19]

Miglioramenti - LFSR

Restano però alcune problematiche:

- Grain necessita di una chiave di 128bit
A tal fine è possibile caricare un valore randomico a seguire della chiave nel cifrario per garantire più entropia dei processi di autenticazione.
- Alternativamente sarebbe necessario aumentare la lunghezza della chiave.
Per fare ciò sarebbe poi necessario modificare la struttura di memoria oppure ridurre lo spazio consentito ai dati del tag.

2022-03-24



Miglioramenti - LFSR

Restano però alcune problematiche:

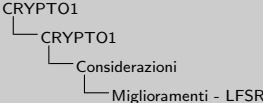
- Grain necessita di una chiave di 128bit
A tal fine è possibile caricare un valore randomico a seguire della chiave nel cifrario per garantire più entropia dei processi di autenticazione.
- Alternativamente sarebbe necessario aumentare la lunghezza della chiave.
Per fare ciò sarebbe poi necessario modificare la struttura di memoria oppure ridurre lo spazio consentito ai dati del tag.

Miglioramenti - LFSR

Restano però alcune problematiche:

- Grain necessita di una chiave di 128bit
A tal fine è possibile caricare un valore randomico a seguire della chiave nel cifrario per garantire più entropia dei processi di autenticazione.
- Alternativamente sarebbe necessario aumentare la lunghezza della chiave.
Per fare ciò sarebbe poi necessario modificare la struttura di memoria oppure ridurre lo spazio consentito ai dati del tag.

2022-03-24



In ogni caso la lunghezza della chiave di 48 bit è da considerarsi non siura, tanto quanto la tecnologi hardware.

Restano però alcune problematiche:

- Grain necessita di una chiave di 128bit
A tal fine è possibile caricare un valore randomico a seguire della chiave nel cifrario per garantire più entropia dei processi di autenticazione.
- Alternativamente sarebbe necessario aumentare la lunghezza della chiave.
Per fare ciò sarebbe poi necessario modificare la struttura di memoria oppure ridurre lo spazio consentito ai dati del tag.

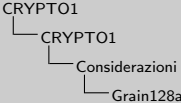
Grain128a i

Vantaggi

- Crittograficamente sicuro
- Può includere un MAC

Grain utilizza chiavi da 128 bit e IV da 96 bit, mentre la struttura interna è composta da un LFSR da 128 bit e da un NLFSR anch’esso da 128 bit

2022-03-24



Grain128a i

Vantaggi

- Crittograficamente sicuro
- Può includere un MAC

Grain utilizza chiavi da 128 bit e IV da 96 bit, mentre la struttura interna è composta da un LFSR da 128 bit e da un NLFSR anch’esso da 128 bit

Grain128a ii

- 1 Sommario
- 2 ISO14443
- 3 Cifrari di flusso
- 4 CRYPTO1
 - Storia
 - Vulnerabilità
 - Reverse Engineering
 - Attacchi
 - Considerazioni
 - RNG
 - TRNG
 - LFSR
 - Grain128
 - Successori
- 5 Bibliografia

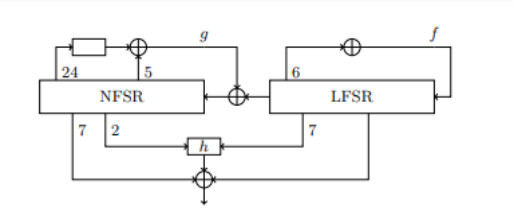


Figura 16: Schema logico di Grain128/Grain128a

All'interno di Grain è possibile trovare due registri a scorrimento, uno lineare e uno non lineare.

In aggiunta h è una funzione non lineare che contribuisce all'output.

2022-03-24

- CRYPTO1
 - Considerazioni
 - Grain128a

Grain128a ii

Figura 16: Schema logico di Grain128/Grain128a

All'interno di Grain è possibile trovare due registri a scorrimento, uno lineare e uno non lineare.

In aggiunta h è una funzione non lineare che contribuisce all'output.

Grain128a iii	
1	Sommario
2	ISO14443
3	Cifrari di flusso
4	CRYPTO1
	Storia
	Vulnerabilità
	Reverse Engineering
	Attacchi
	Considerazioni
	RNG
	TRNG
	LFSR
	Grain128
	Successori
5	Bibliografia

2022-03-24	CRYPTO1
	CRYPTO1
	Considerazioni
	Grain128a iii

Per inizializzare il cifrario, una chiave da 128bit e un IV da 96 bit vengono inseriti nell'NLFSR e nel LFSR rispettivamente, completando l'LFSR con una costante.

Per inizializzare il cifrario, una chiave da 128bit e un IV da 96 bit vengono inseriti nell'NLFSR e nel LFSR rispettivamente, completando l'LFSR con una costante.

Grain128a - Autenticazione i

Per autenticare il tag è possibile utilizzare il cifrario Grain128a:

Il keystream è dato da y_{64+2n} ovvero dai bit dispari dell'output scartando i primi 64.

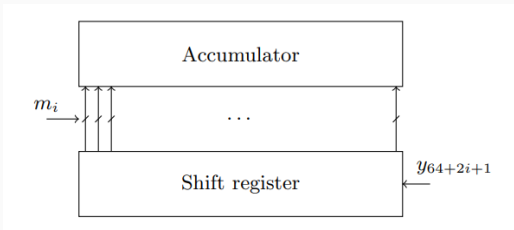


Figura 17: Schema logico dell'autenticazione utilizzata da Grain128a

2022-03-24

- CRYPTO1
 - Considerazioni
 - Grain128a - Autenticazione i

Grain128a - Autenticazione i

Per autenticare il tag è possibile utilizzare il cifrario Grain128a:
Il keystream è dato da y_{64+2n} ovvero dai bit dispari dell'output scartando i primi 64.

Figura 17: Schema logico dell'autenticazione utilizzata da Grain128a

Grain128a - Autenticazione ii

Assumiamo di avere un messaggio $\bar{m} = m_0, m_1, \dots, m_{L-1}$ di lunghezza L .

Per garantire che \bar{m} e $\bar{m}||0$ abbiano risultato diverso (attacchi di tipo extension) poniamo $m_L = 1$

Inizializzazione

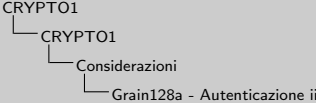
Il registro accumulatore viene inizializzato con i primi 32 bit del keystream, il registro a scorrimento viene inizializzato con i successivi 32.

Autenticazione

Codificando il messaggio, il registro a scorrimento sarà aggiornato ogni due bit del keystream ($r_{i+31}=y_{64+2i+1}$) mentre l'accumulatore sarà aggiornato secondo

$$a_{i+1} = a_i \oplus m \cdot r$$

2022-03-24



Assumiamo di avere un messaggio $\bar{m} = m_0, m_1, \dots, m_{L-1}$ di lunghezza L .
Per garantire che \bar{m} e $\bar{m}||0$ abbiano risultato diverso (attacchi di tipo extension) poniamo $m_L = 1$

Inizializzazione

Il registro accumulatore viene inizializzato con i primi 32 bit del keystream, il registro a scorrimento viene inizializzato con i successivi 32.

Autenticazione

Codificando il messaggio, il registro a scorrimento sarà aggiornato ogni due bit del keystream ($r_{i+31}=y_{64+2i+1}$) mentre l'accumulatore sarà aggiornato secondo

$$a_{i+1} = a_i \oplus m \cdot r$$

Grain128a - Autenticazione ii

Assumiamo di avere un messaggio $\bar{m} = m_0, m_1, \dots, m_{L-1}$ di lunghezza L .

Per garantire che \bar{m} e $\bar{m}||0$ abbiano risultato diverso (attacchi di tipo extension) poniamo $m_L = 1$

Inizializzazione

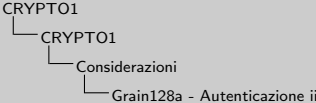
Il registro accumulatore viene inizializzato con i primi 32 bit del keystream, il registro a scorrimento viene inizializzato con i successivi 32.

Autenticazione

Codificando il messaggio, il registro a scorrimento sarà aggiornato ogni due bit del keystream ($r_{i+31}=y_{64+2i+1}$) mentre l'accumulatore sarà aggiornato secondo

$$a_{i+1} = a_i \oplus m \cdot r$$

2022-03-24



Grain128a - Autenticazione ii

Assumiamo di avere un messaggio $\bar{m} = m_0, m_1, \dots, m_{L-1}$ di lunghezza L .
Per garantire che \bar{m} e $\bar{m}||0$ abbiano risultato diverso (attacchi di tipo extension) poniamo $m_L = 1$

Inizializzazione

Il registro accumulatore viene inizializzato con i primi 32 bit del keystream, il registro a scorrimento viene inizializzato con i successivi 32.

Autenticazione

Codificando il messaggio, il registro a scorrimento sarà aggiornato ogni due bit del keystream ($r_{i+31}=y_{64+2i+1}$) mentre l'accumulatore sarà aggiornato secondo

$$a_{i+1} = a_i \oplus m \cdot r$$

Grain128a - Autenticazione ii

Assumiamo di avere un messaggio $\bar{m} = m_0, m_1, \dots, m_{L-1}$ di lunghezza L .

Per garantire che \bar{m} e $\bar{m}||0$ abbiano risultato diverso (attacchi di tipo extension) poniamo $m_L = 1$

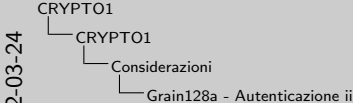
Inizializzazione

Il registro accumulatore viene inizializzato con i primi 32 bit del keystream, il registro a scorrimento viene inizializzato con i successivi 32.

Autenticazione

Codificando il messaggio, il registro a scorrimento sarà aggiornato ogni due bit del keystream ($r_{i+31}=y_{64+2i+1}$) mentre l'accumulatore sarà aggiornato secondo

$$a_{i+1} = a_i \oplus m \cdot r$$



Il risultato dell'autenticazione è quindi il valore finale dell'accumulatore che sarà uguale sia per la cifratura che la decifratura del messaggio.

Grain128a - Autenticazione ii

Assumiamo di avere un messaggio $\bar{m} = m_0, m_1, \dots, m_{L-1}$ di lunghezza L .

Per garantire che \bar{m} e $\bar{m}||0$ abbiano risultato diverso (attacchi di tipo extension) poniamo $m_L = 1$

Inizializzazione

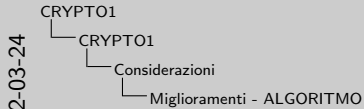
Il registro accumulatore viene inizializzato con i primi 32 bit del keystream, il registro a scorrimento viene inizializzato con i successivi 32.

Autenticazione

Codificando il messaggio, il registro a scorrimento sarà aggiornato ogni due bit del keystream ($r_{i+31}=y_{64+2i+1}$) mentre l'accumulatore sarà aggiornato secondo

$$a_{i+1} = a_i \oplus m \cdot r$$

Miglioramenti - ALGORITMO	
1	Sommario
2	ISO14443
3	Cifrari di flusso
4	CRYPTO1
	Storia
	Vulnerabilità
	Reverse Engineering
	Attacchi
	Considerazioni
	RNG
	TRNG
	LFSR
	Grain128
	Successori
5	Bibliografia



2022-03-24

Miglioramenti - ALGORITMO

Una soluzione più drastica è rappresentata dal cambio del circuito di cifratura, passando a un'implementazione AES a basso costo.[FWR05]

In questo caso resta la problematica della gestione di chiavi a 128 bit.

Successori

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse
Engineering

Attacchi

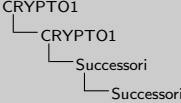
Considerazioni

Successori

5 Bibliogra-
fia

- MIFARE Plus: Drop-in replacement con algoritmo modificato AES-128. Data la compatibilità con sistemi MIFARE Classic senza supporto ad AES, presenta ancora tutte le vulnerabilità precedentemente discusse
- MIFARE DesFire: Utilizza AES e DES/3DES per garantire la sicurezza ma esse sono notevolmente più costose: utilizzano un microcontrollore sul quale è possibile eseguire un sistema operativo. Permettono quindi di essere usate come SecureElements

2022-03-24



Successori

- MIFARE Plus: Drop-in replacement con algoritmo modificato AES-128. Data la compatibilità con sistemi MIFARE Classic senza supporto ad AES, presenta ancora tutte le vulnerabilità precedentemente discusse
- MIFARE DesFire: Utilizza AES e DES/3DES per garantire la sicurezza ma esse sono notevolmente più costose: utilizzano un microcontrollore sul quale è possibile eseguire un sistema operativo. Permettono quindi di essere usate come SecureElements

Successori

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

Storia

Vulnerabilità

Reverse
Engineering

Attacchi

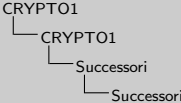
Considerazioni

Successori

5 Bibliogra-
fia

- MIFARE Plus: Drop-in replacement con algoritmo modificato AES-128. Data la compatibilità con sistemi MIFARE Classic senza supporto ad AES, presenta ancora tutte le vulnerabilità precedentemente discusse
- MIFARE DesFire: Utilizza AES e DES/3DES per garantire la sicurezza ma esse sono notevolmente più costose: utilizzano un microcontrollore sul quale è possibile eseguire un sistema operativo. Permettono quindi di essere usate come SecureElements

2022-03-24



Successori

- MIFARE Plus: Drop-in replacement con algoritmo modificato AES-128. Data la compatibilità con sistemi MIFARE Classic senza supporto ad AES, presenta ancora tutte le vulnerabilità precedentemente discusse
- MIFARE DesFire: Utilizza AES e DES/3DES per garantire la sicurezza ma esse sono notevolmente più costose: utilizzano un microcontrollore sul quale è possibile eseguire un sistema operativo. Permettono quindi di essere usate come SecureElements

Bibliografia

2022-03-24

CRYPTO1
Bibliografia

Bibliografia

Bibliografia i

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

5 Bibliogra-
fia

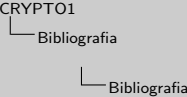


Nicolas T Courtois, Karsten Nohl, and Sean O’Neil.
Algebraic attacks on the crypto-1 stream cipher in mifare classic and oyster cards.
Cryptology ePrint Archive, 2008.



Vedat Coskun, Busra Ozdenizci, and Kerem Ok.
A survey on near field communication (nfc) technology.
Wireless personal communications, 71(3):2259–2294, 2013.

2022-03-24



Bibliografia i

-  Nicolas T Courtois, Karsten Nohl, and Sean O’Neil.
Algebraic attacks on the crypto-1 stream cipher in mifare classic and oyster cards.
Cryptology ePrint Archive, 2008.
-  Vedat Coskun, Busra Ozdenizci, and Kerem Ok.
A survey on near field communication (nfc) technology.
Wireless personal communications, 71(3):2259–2294, 2013.

Bibliografia ii

- 1 Sommario
- 2 ISO14443
- 3 Cifrari di flusso
- 4 CRYPTO1
- 5 Bibliografia



Nicolas T. Courtois.

The dark side of security by obscurity - and cloning mifare classic rail and building passes, anywhere, anytime.

IACR Cryptol. ePrint Arch., 2009:137, 2009.



Igor Fermevc and Saša Adamović.

Low-cost portable trng, implementation and evaluation.

Serbian Journal of Electrical Engineering, 13(3):361–368, 2016.

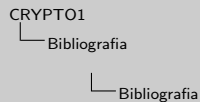


Martin Feldhofer, Johannes Wolkerstorfer, and Vincent Rijmen.

Aes implementation on a grain of sand.

IEE Proceedings-Information Security, 152(1):13–20, 2005.

2022-03-24




Bibliografia ii


 Nicolas T. Courtois.
The dark side of security by obscurity - and cloning mifare classic rail and building passes, anywhere, anytime.
IACR Cryptol. ePrint Arch., 2009:137, 2009.

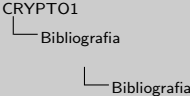
 Igor Fermevc and Saša Adamović.
Low-cost portable trng, implementation and evaluation.
Serbian Journal of Electrical Engineering, 13(3):361–368, 2016.

 Martin Feldhofer, Johannes Wolkerstorfer, and Vincent Rijmen.
Aes implementation on a grain of sand.
IEE Proceedings-Information Security, 152(1):13–20, 2005.


Bibliografia iii


 Martin ? gren, Martin Hell, Thomas Johansson, and Willi Meier.
Grain-128a: a new version of grain-128 with optional authentication.
International Journal of Wireless and Mobile Computing, 5(1):48–59, 2011.

 Flavio D Garcia, Gerhard de Koning Gans, Ruben Muijrsers, Peter van Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs.
Dismantling mifare classic.
In *European symposium on research in computer security*, pages 97–114.
Springer, 2008.



Bibliografia iii

 Martin ? gren, Martin Hell, Thomas Johansson, and Willi Meier.
Grain-128a: a new version of grain-128 with optional authentication.
International Journal of Wireless and Mobile Computing, 5(1):48–59, 2011.

 Flavio D Garcia, Gerhard de Koning Gans, Ruben Muijrsers, Peter van Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs.
Dismantling mifare classic.
In *European symposium on research in computer security*, pages 97–114.
Springer, 2008.

1 Sommario

2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

5 Bibliogra-
fia

Bibliografia iv



Kuo Huang.
Secured rfid mutual authentication scheme for mifare systems.
International Journal of Network Security and Its Applications, 4:17–31, 11
2012.

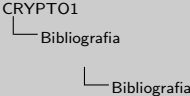


Texas Instruments.
Iso/nfc standards and specifications overview.
Texas: Texas Instruments, 2014.





Karsten Nohl, David Evans, Starbug Starbug, and Henryk Plötz.
Reverse-engineering a cryptographic rfid tag.
In *USENIX security symposium*, volume 28, 2008.


2022-03-24



Bibliografia iv

 Kuo Huang.
Secured rfid mutual authentication scheme for mifare systems.
International Journal of Network Security and Its Applications, 4:17–31, 11
2012.

 Texas Instruments.
Iso/nfc standards and specifications overview.
Texas: Texas Instruments, 2014.

 Karsten Nohl, David Evans, Starbug Starbug, and Henryk Plötz.
Reverse-engineering a cryptographic rfid tag.
In *USENIX security symposium*, volume 28, 2008.

1 Sommario


2 ISO14443

3 Cifrari di
flusso

4 CRYPTO1

5 Bibliogra-
fia

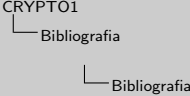
Bibliografia v

 NXP.
An10927: Mifare product and handling of uids.
NXP Applications notes, 2018.


 NXP.
Mifare classic ev1 1k - mainstream contactless smart card ic for fast and easy solution development.
NXP Datasheets, 2018.


 Jonathan Sönnerup, Martin Hell, Mattias Sönnerup, and Ripudaman Khattar.
Efficient hardware implementations of grain-128aead.
In *International Conference on Cryptology in India*, pages 495–513. Springer, 2019.


2022-03-24



Bibliografia v

 NXP.
An10927: Mifare product and handling of uids.
NXP Applications notes, 2018.

 NXP.
Mifare classic ev1 1k - mainstream contactless smart card ic for fast and easy solution development.
NXP Datasheets, 2018.

 Jonathan Sönnerup, Martin Hell, Mattias Sönnerup, and Ripudaman Khattar.
Efficient hardware implementations of grain-128aead.
In *International Conference on Cryptology in India*, pages 495–513. Springer, 2019.

Bibliografia vi

1 Sommario

2 ISO14443

3 Cifrari di flusso

4 CRYPTO1

5 Bibliografia



Karen Scarfone, Wayne Jansen, Miles Tracy, et al.

Guide to general server security.

NIST Special Publication, 800(123), 2008.

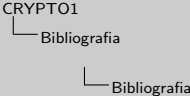


Cihangir Tezcan.


Brute force cryptanalysis of mifare classic cards on gpu.

In *International Conference on Information Systems Security and Privacy*, volume 2, pages 524–528. SciTePress, 2017.

2022-03-24




Bibliografia vi



Karen Scarfone, Wayne Jansen, Miles Tracy, et al.

Guide to general server security.

NIST Special Publication, 800(123), 2008.



Cihangir Tezcan.

Brute force cryptanalysis of mifare classic cards on gpu.

In *International Conference on Information Systems Security and Privacy*, volume 2, pages 524–528. SciTePress, 2017.

1 Sommario

2 ISO14443

3 Cifrari di
flusso

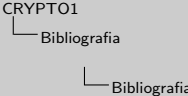
4 CRYPTO1

5 Bibliogra-
fia



Roel Verdult, Flavio D Garcia, and Josep Balasch.
Gone in 360 seconds: Hijacking with hitag2.
In *21st USENIX Security Symposium (USENIX Security 12)*, pages 237–252,
2012.

2022-03-24



Bibliografia vii

Roel Verdult, Flavio D Garcia, and Josep Balasch.
Gone in 360 seconds: Hijacking with hitag2.
In *21st USENIX Security Symposium (USENIX Security 12)*, pages 237–252,
2012.