AfricaCrypt 2008
Casablanca, Morocco, JUNE 11-14; 2008

Weaknesses in a Recent Ultra-Lightweight Rfid Authentication Protocol

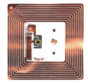
by

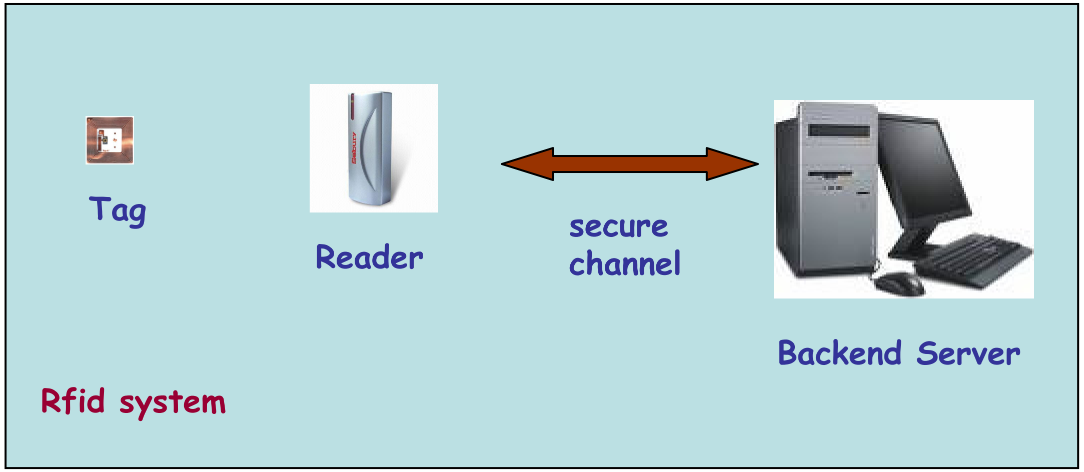
Paolo D'Arco and Alfredo De Santis

University of Salerno
Italy

Rfid Technology

- Automatic object identification
- Tag: microchip equipped with an antenna
- A Reader gets data from the Tag





Applications

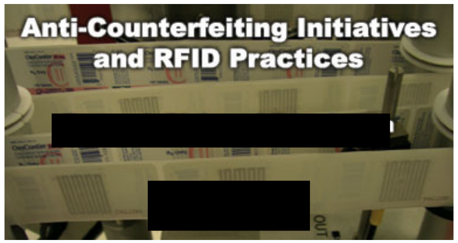
access control

secure e-passport

automatic pay-toll

inventory control

pet identification



tags for medicines

supermarket checkout counters

Authentication Protocols

The Reader need to be sure the tag is not counterfeit

" … anti-counterfeiting tags for medicines"

anti-counterfeiting tags for medicines

The Tag need to be sure the Reader is a legal one

" … supermarket checkout counters"

supermarket checkout counters

Passive Tags

- cheap
- low memory storage
- simple circuitry
- no power source

… lightweight authentication protocols should be provided …

State of Art.

HB-like family (based on the LPN problem)

Squash-based authentication protocol (based on Rabin PK scheme)

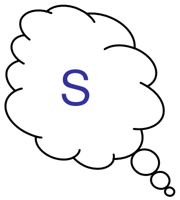
A different approach

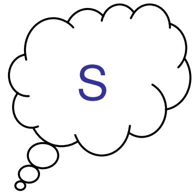
- Ultra-lightweight authentication protocol
- Simple bitwise operations (AND, OR, XOR, Rot, $+_{\text{mod } n}$, ...)
- Security analysis: intuitions and reasonable arguments

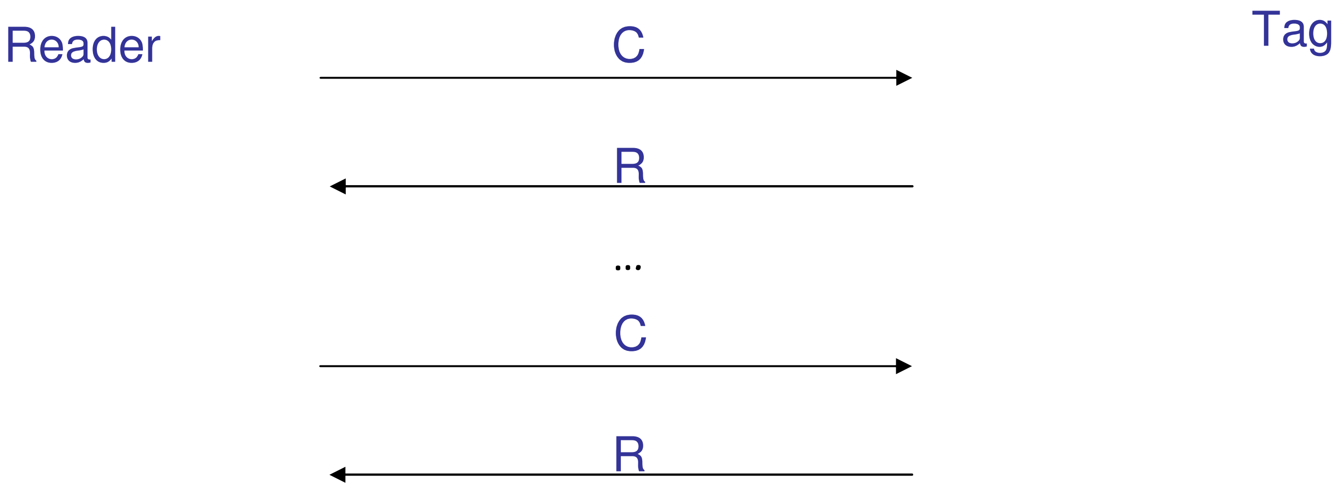
SASI

Strong **A**uthentication and **S**trong **I**ntegrity

[Chien07, IEEE Transactions on Dependable and Secure Computing, Vol.4, N. 4, Oct-Dec 2007]







A challenge-response protocol

Bitwise operations: AND, OR, XOR, Rot, $+_{mod\ n}$

SASI

Reader

ID
IDS, K_1, K_2

Tag

ID
IDS, K_1, K_2

Reader:

$Choose\ n_1, n_2$

$A = IDS \oplus K_1 \oplus n_1$

$B = (IDS \vee K_2) + n_2$

$K_1^{'} = Rot(K_1 \oplus n_2, K_1)$

$K_2^{'} = Rot(K_2 \oplus n_1, K_2)$

$C = (K_1 \oplus K_2^{'}) + (K_1^{'} \oplus K_2)$

$D_R = (K_1^{'} + ID) \oplus ((K_1 \oplus K_2) \vee K_1^{'})$

$If\ D_R = D\ then\ accept$

Messages:

Hello

IDS

A,B,C

D

Tag:

$Extract\ n_1, n_2$

$K_1^{'} = Rot(K_1 \oplus n_2, K_1)$

$K_2^{'} = Rot(K_2 \oplus n_1, K_2)$

$C_T = (K_1 \oplus K_2^{'}) + (K_1^{'} \oplus K_2)$

$If\ C_T = C\ then\ accept$

$Send$

$D = (K_1^{'} + ID) \oplus ((K_1 \oplus K_2) \vee K_1^{'})$

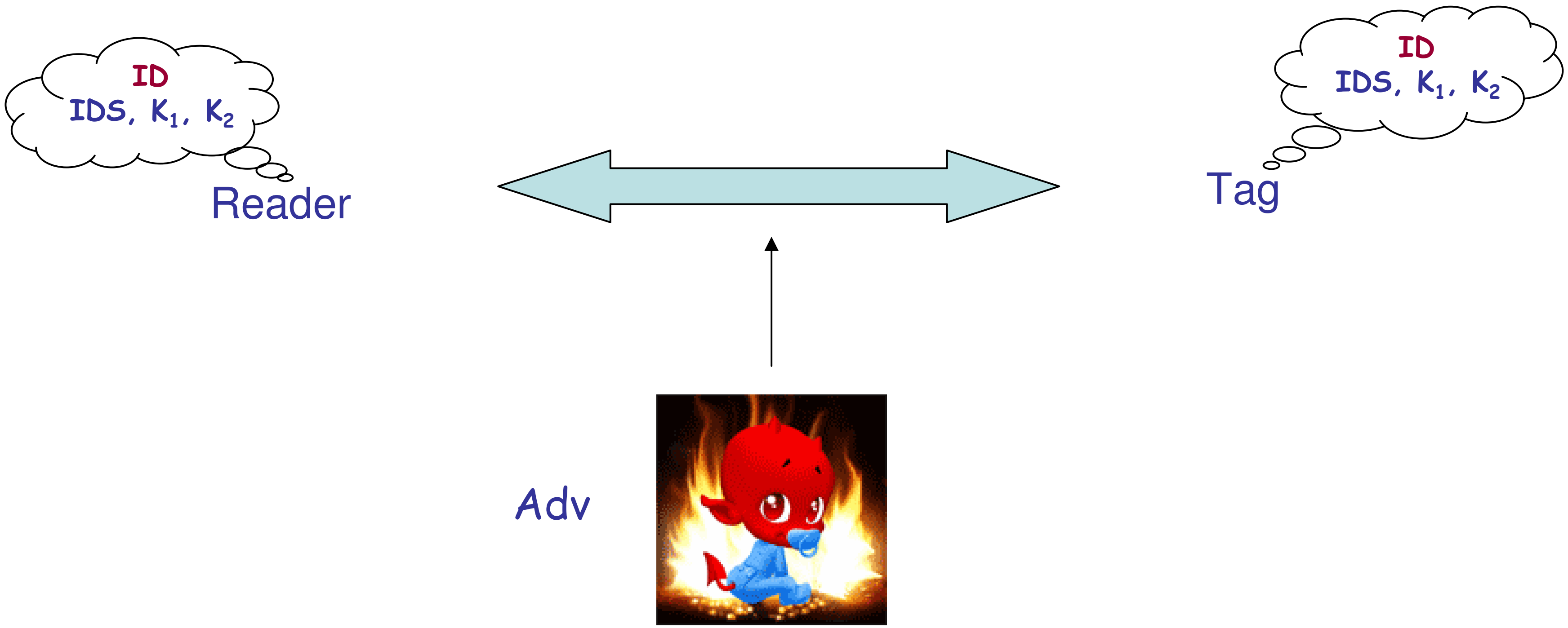
Update

IDS', K'_1, K'_2

Update

IDS', K'_1, K'_2
IDS, K_1, K_2

Attack Model



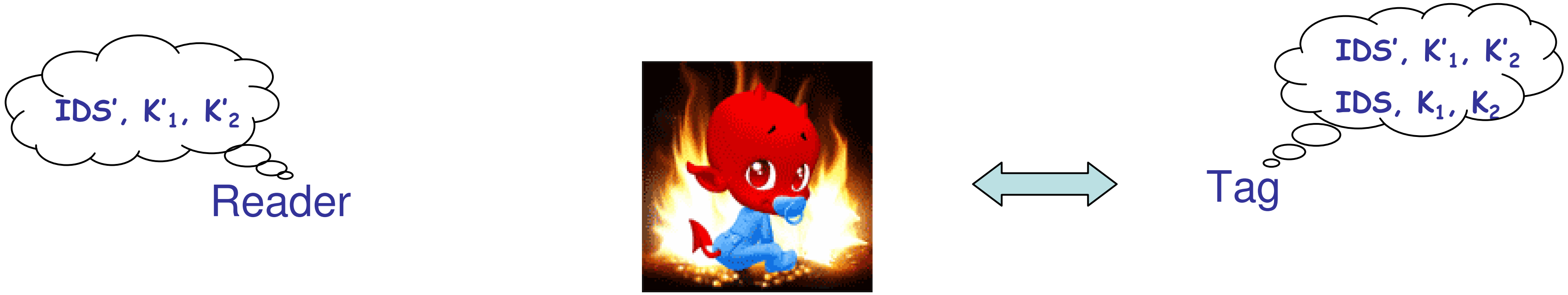
- **Eavesdrops**
- **Sends/Intercepts msgs**

Contribution of this paper

- De-synchronisation attack
  - Tag and Reader do not share a tuple any more
- Identity Disclosure Attack
  - Adv recovers the tag ID
- Full Disclosure Attack
  - Adv computes all secret data of the tag

Average number of trials	48.5
Average number of trials	241
Average number of trials	242

De-synchronisation: Idea



Looks at an execution of the authentication protocol

- **Reset stage**: Adv resets the Tag to the *same state* in which it was before executing the authentication protocol with the Reader

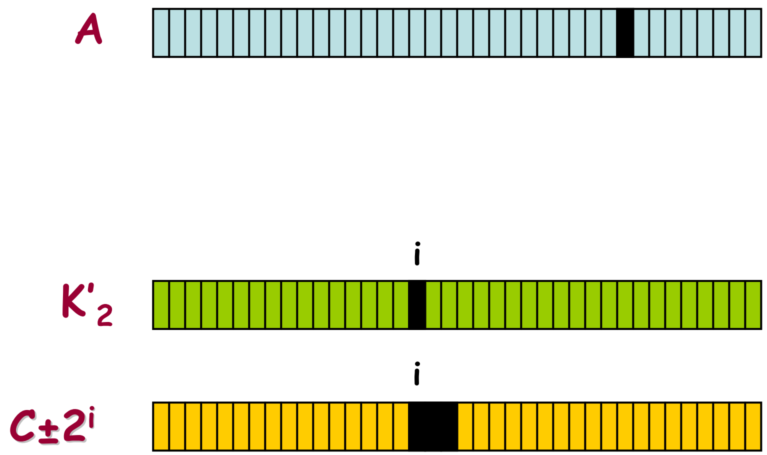
- **Trial stage**: Adv, using A, B and C, construct *a new triple* A', B, C' which could be accepted by the Tag

How to construct a new triple?

$$\text{Choose } n_1, n_2$$
$$A = IDS \oplus K_1 \oplus n_1$$
$$B = (IDS \vee K_2) + n_2$$

$$K_1' = Rot(K_1 \oplus n_2, K_1)$$
$$K_2' = Rot(K_2 \oplus n_1, K_2)$$

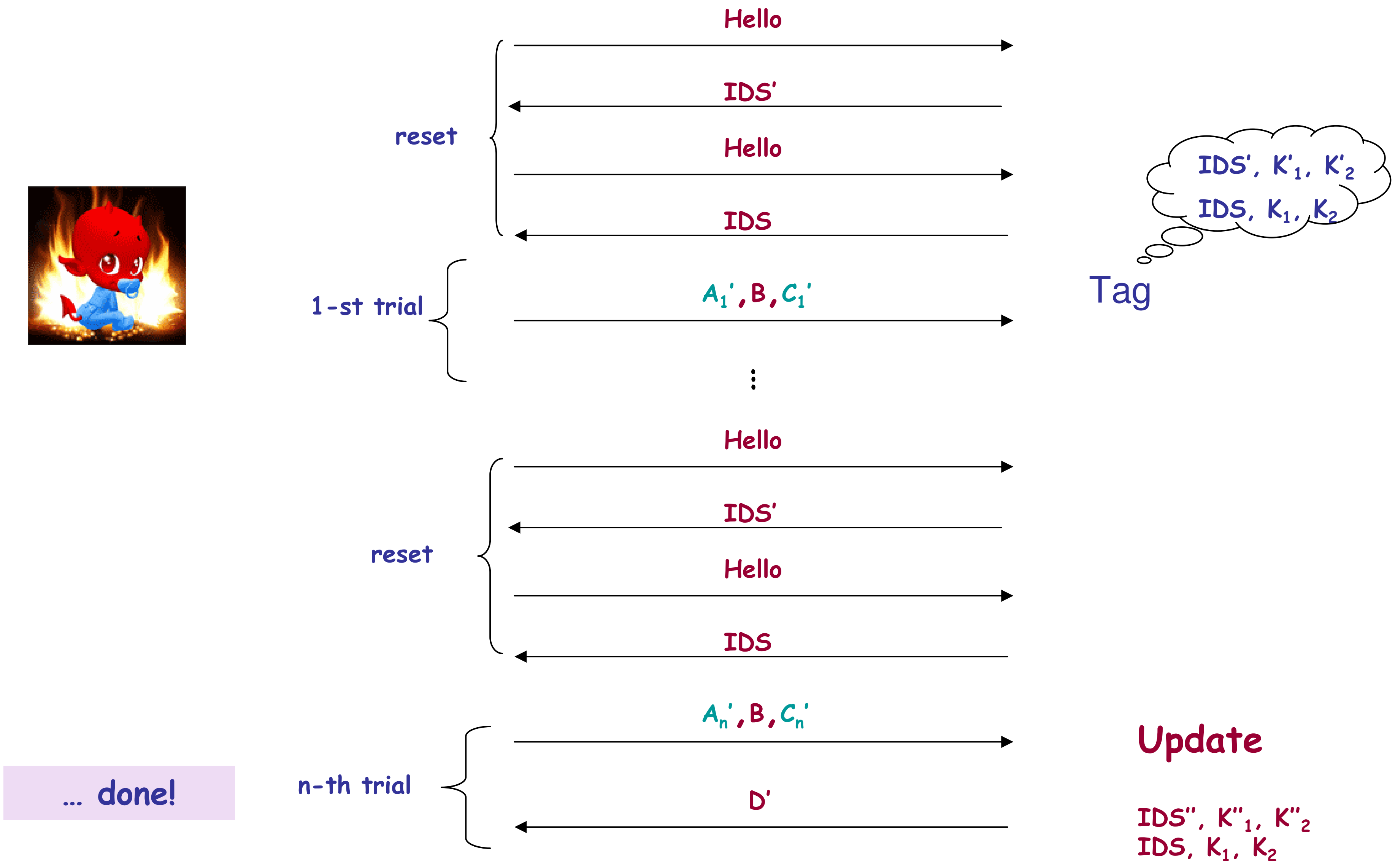
$$C = (K_1 \oplus K_2') + (K_1' \oplus K_2)$$



Flipping a bit in *A* implies changing a bit in K'_2.

The value of *C* becomes $C \pm 2^i$

De-synchronisation



De-synchronisation

IDS', K'_1, K'_2

Reader

IDS', K'_1, K'_2
IDS, K_1, K_2

Tag

Adv, interacts with the Tag by sending tuples A', B, C', as long as the Tag accepts. When Adv gets D' at the n-th trial, Tag and Reader have been de-synchronised

Update

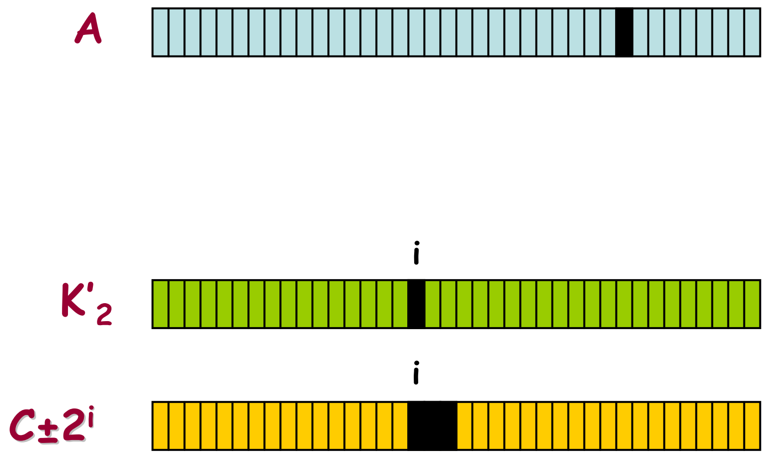
IDS', K'_1, K'_2

no shared tuple

IDS'', K''_1, K''_2
IDS, K_1, K_2

De-synchronisation

A note: when Adv succedes
in de-synchronising Reader and Tag,
then Adv knows also
the amount of
the Rot operation for K'_2

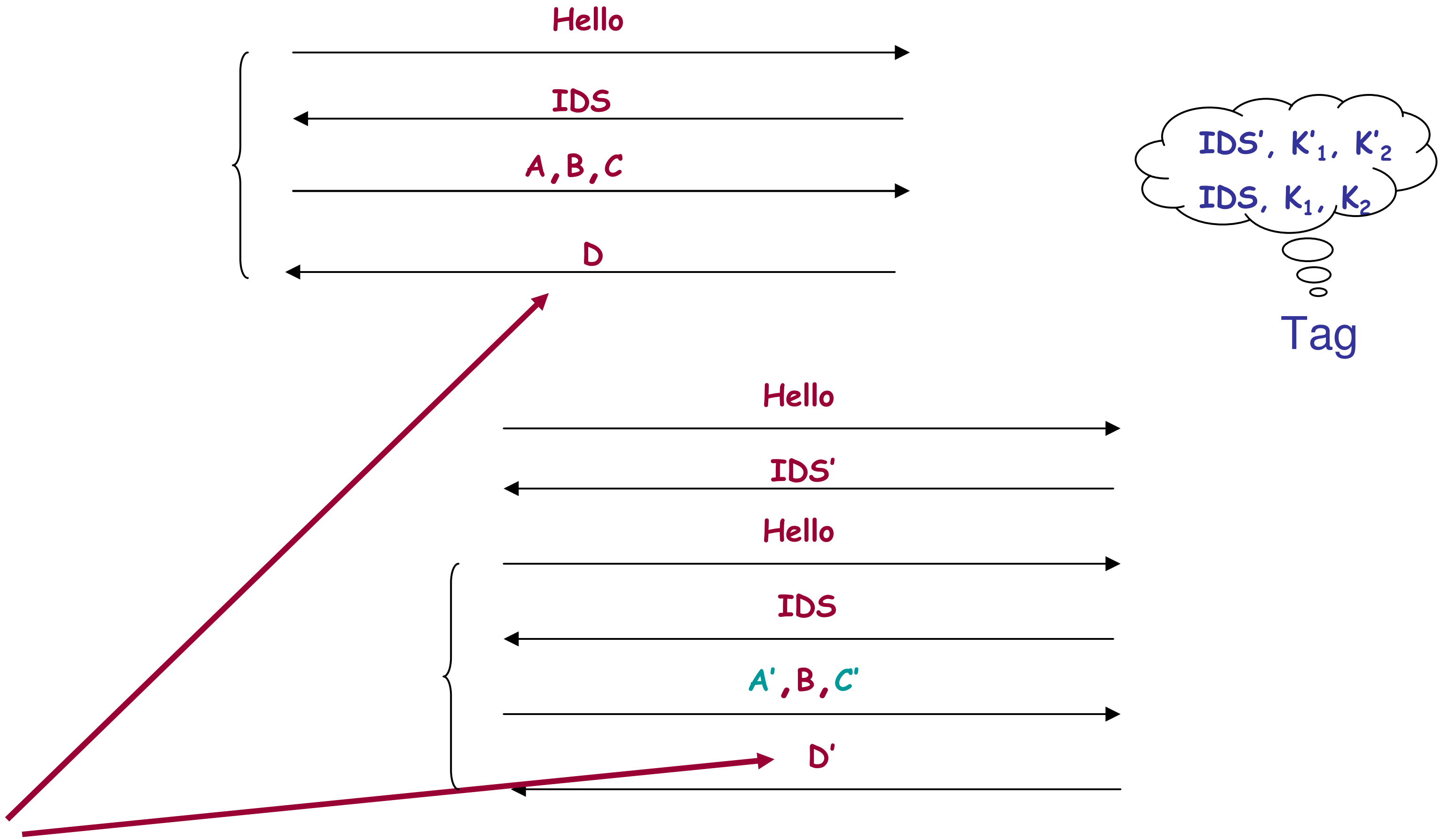


Adv can

• control K'_2

• construct a new triple A', B, C' which is accepted on average after 1.5 trials

Identity Disclosure: Idea



… looking at differences between Tag replies, Adv gets information about the bits of the static ID …

Identity Disclosure: 1-st bit

… looking at differences between Tag replies …

$$D = (K_2' + ID) \oplus ((K_1 \oplus K_2) \vee K_1'$$

$$\oplus$$

$$D' = (\overline{K'}_2 + ID) \oplus ((K_1 \oplus K_2) \vee K_1'$$

$$D = K_2'[95] + ID[95], \quad \ldots \quad K_2'[1] + ID[1], K_2'[0] + ID[0]$$

$$\oplus$$

$$D' = K_2'[95] + ID[95], \quad \ldots \quad K_2'[1] + ID[1], \overline{K'}_2[0] + ID[0]$$

$$= \qquad 0/1 \qquad 1$$

Adv has control over K'_2 . Forcing $\overline{K'_2[0]}$ to be different from $K'_2[0]$ (while all other entries are equal) it holds that, if

- $D[1] \otimes D'[1] = 0$ ⟶ $ID[0]=0$
- $D[1] \otimes D'[1] = 1$ ⟶ $ID[0]=1$

Identity Disclosure

… to recover the other bits of ID, Adv still looks at the differences, i.e., for i=1, …, 94,

$$ID[i] = D[i+1] \otimes D^{i}[i+1]$$

… but Adv needs a pre-processing stage in order to avoid *carry generation*

$$(K'_2[i]+ID[i]+c_i) \oplus (K'_2[i]+ID[i]+c_i)$$

Adv manipulates K'_2 in such a way that, for i=1, …, 95,

$K'_2[i]$ is different from ID[i])

Adv can do it, interacting with the Tag, efficiently!

Identity Disclosure: Preprocessing

Adv's Computation.

1. Constructs and sends to the Tag a new sequence $\mathbf{A_r}||\mathbf{B}||\mathbf{C_r}$, modifying $\mathbf{A}$ and $\mathbf{C}$, in order to flip the r-th bit of $\overline{K}_2$.
2. The Tag replies with a value $\mathbf{D_r}$.
3. Then, *Adv* sends to the Tag a new sequence $\mathbf{A_r}^0||\mathbf{B}||\mathbf{C_r}^0$, constructed from $\mathbf{A_r}||\mathbf{B}||\mathbf{C_r}$, in order to flip the first bit of the new $\overline{K}_2$, i.e., $\overline{K}_2[0]$.
4. The tag replies with $\mathbf{D_r}^0$.
5. *Adv* computes $P = \mathbf{D_r} \oplus \mathbf{D_r}^0 = 0^{96-t}1^t$, where $t > r$. If $t = 96$, then *Adv* has finished; otherwise, *Adv* repeats the procedure working on the t-th bit, that is, setting $r = t$ and $\mathbf{A}||\mathbf{B}||\mathbf{C} = \mathbf{A_r}||\mathbf{B}||\mathbf{C_r}$.

Figure 4: Pre-processing for the identity disclosure attack.

Full Disclosure

Adv works as follows

- eavesdrops an execution of the protocol
- ID disclosure attack
- resets the tag to the previous state
- eavesdrops other two executions of the protocol
- computes the secret keys

Conclusions

- We have proposed three efficient attacks against SASI
    - De-synchronisation
    - Identity Disclosure
    - Full Disclosure
    - Implementation and testing

- Sound security arguments should be used to support cryptographic protocol design

Related Work

- **On the Security of Chien's Ultralightweight RFID Authentication Protocol,** *H-M. Sun, W-C. Ting, and K-H. Wang* (eprint archive, N. 83, Feb. 2008)

two de-synchronisation attacks

- **Cryptanalysis of the SASI Ultralightweight RFID Authentication Protocol,** *J. C. Hernandez-* Castro, J. M. E. Tapiador, P. Peris-Lopez, and J-J. Quisquater (private communication, Jun 2008, submitted for publication)

ID disclosure–passive attack