QSNFC: Quick and Secured Near Field Communication for the Internet of Things

RFID'18

Thomas Ulz, Graz UT
Thomas Pieber, Graz UT
Christian Steger, Graz UT
Sarah Haas, Infineon Austria
Rainer Matischek, Infineon Austria

Outline

1. Introduction & Motivation
2. QSNFC Protocol
    1. System Model & Protocol Stack
    2. Connection Establishment
    3. Connection Teardown & Cache Management
3. Evaluation
    1. Example Use-Cases
    2. Security Analysis
    3. Overhead

Introduction & Motivation

WYASP

Why yet another security protocol?

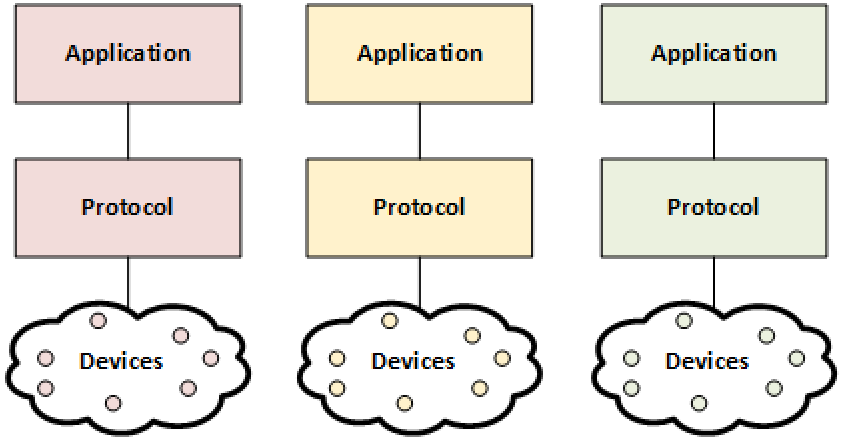
Introduction & Motivation

- As mentioned in yesterday‘s RFID security tutorial
- Asymmetric cryptography
  - High hardware complexity
  - Power consumption high
  - Throughput low
- Symmetric cryptography
  - Good solution for constrained systems such as RFIDs
  - BUT: key distribution problem

- Same problem in other domains: Internet, Internet of Things, …
  - Many security standards

Introduction & Motivation

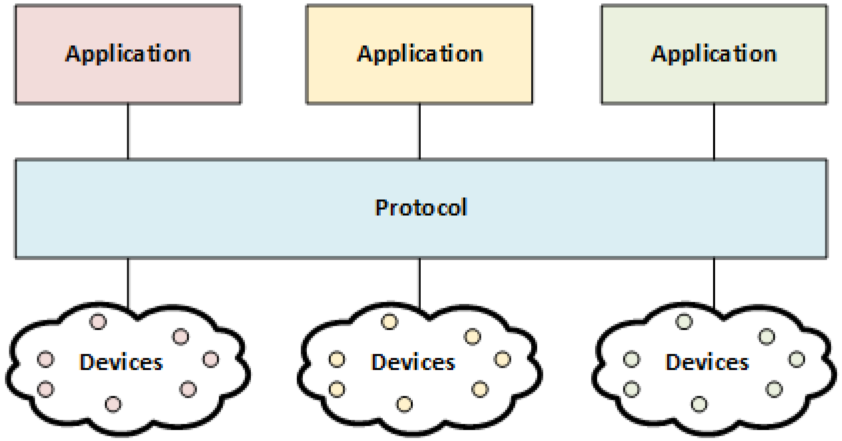
- NFC security standards exist
  - ECMA-385 NFC-SEC: shared secrets for NFCIP-1
- Many NFC protocols that claim to be secured
- Even many initiatives and standards from industry
  - PCI / DSS: payment card industry data security standard
  - EMV / EMV contactless: europay, mastercard, visa
  - CIPURSE: secured fare collection



- However, all these protocols are tailored for one specific domain
  - Payment, fare collection, ticketing, access control, …
  - Often proprietary, security hard to validate

Introduction & Motivation

- Internet of Things (IoT)
  - Very large number of devices
  - Rapidly growing
  - Heterogeneous system
- NFC seen as an enabling factor [Al-Fuqaha 2015]
- Trends towards horizontal architecture
- „One-for-all“ protocols
  - Standard for all domains
  - Security: easy to validate



Introduction & Motivation

- QSNFC: Quick and Secured Near Field Communication
- Protocol that relies on standard security primitives
  - Easy to validate
- Based on Transport Layer Security (TLS) and Google QUIC
- Features
  - Device authentication
  - Key agreement process
  - Secured channel
  - Zero round trip time (0-RTT)
- Applicable to any domain!

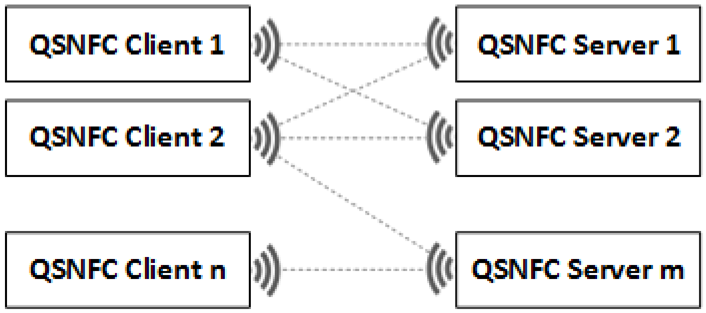
Outline

1. Introduction & Motivation
2. QSNFC Protocol
    1. System Model & Protocol Stack
    2. Connection Establishment
    3. Connection Teardown & Cache Management
3. Evaluation
    1. Example Use-Cases
    2. Security Analysis
    3. Overhead

QSNFC: System Model

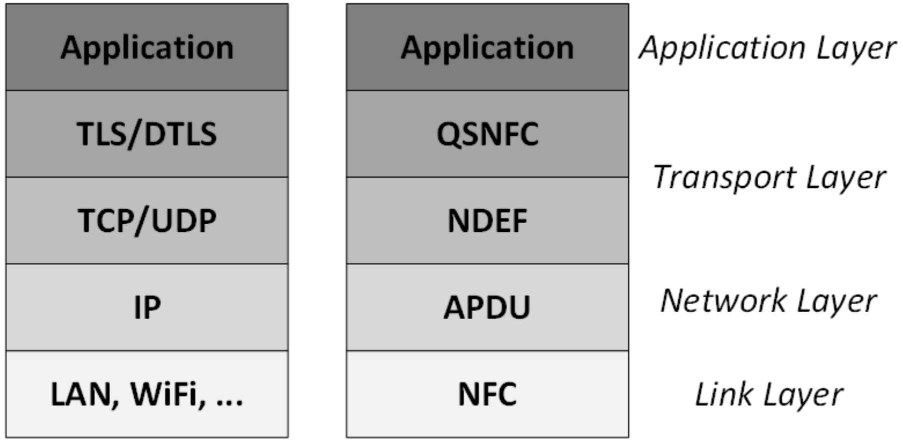
- Based on protocols from the Internet
- There: terms Server and Client
  - Unusual for NFC

- Client
  - Initiates secured connection
  - In NFC terms: active component
- Server
  - Contacted by the client to establish secured connection
  - In NFC terms: passive component



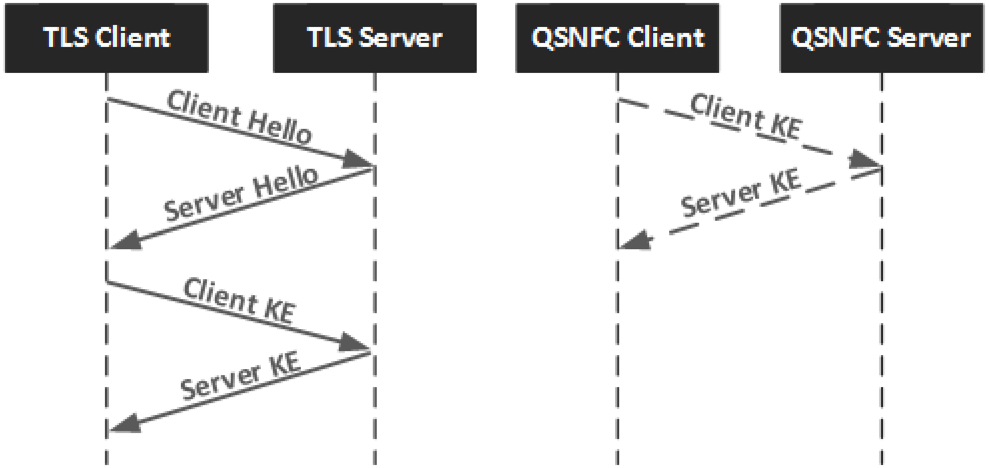
QSNFC: Protocol Stack

- QSNFC handles security relevant features, does not deal with lower layer aspects
  - Packet size
  - Splitting of packets
  - Flags, header fields, …
- QSNFC placed on top of NFC Data Exchange Format (NDEF)
  - Comparable to TLS / DTLS
  - „Transport Layer Security“
- Security features:
  - Transparent for actual application



QSNFC: Connection Establishment

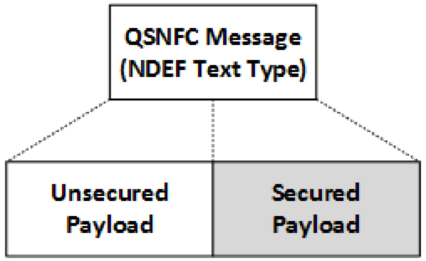
- TLS: 2 round trips needed for connection establishment
- In QSNFC: meets 0-RTT requirement (for recurring connections)



- To achieve this, distinguish between *initial* handshake (HS) and *subsequent* HS

QSNFC: Connection Establishment

- Confidential information is encrypted in every step (AE)
- Initial HS
  - Client and Server communicate with each other for the *first* time
- Client sends so-called *inchoate client hello (CH)*
- Server rejects the CH message (RJ)
- RJ message contains:
  - Server's *long term* public Diffie-Hellman (DH) key
  - Server's certificate for authenticating the server
  - Signature of the long term public DH key
  - Source address token to identify server
- Information cached by client



Type 2 Bit	***LenP*** 2 Byte	***Client ID*** 8 Byte	***Public Key Client*** 16 Byte	***LenE*** 2 Byte
Source Address Token 16 Byte	***Encrypted Payload*** (LenE – 16) Byte			

Type 2 Bit	***LenP*** 2 Byte	***Server ID*** 8 Byte	***Long Term Public Key*** 16 Byte	***Signature*** 8 Byte
Certificate Chain (LenP – 34) Byte	***LenE*** 2 Byte	***Source Address Token*** 16 Byte	***Encrypted Payload*** 0 Byte	

QSNFC: Connection Establishment

- After intial HS, client and server „know each other“
  - Long term public DH key cached
  - Forward secure session keys can be derived using client‘s ephemeral key
  - Client can send *complete* CH, containing client‘s ephemeral public key
- For any subsequent connection establishment
  - Client directly can send complete CH
- Server answers with server hello (SH)
  - Contains server‘s ephemeral public key
  - After this, shared forward secure session key established
- After handshake is complete:
  - Standard data messages

Type 2 Bit	***LenP*** 2 Byte	***Server ID*** 8 Byte	***Public Key Server*** 16 Byte	***LenE*** 2 Byte

Encrypted Payload LenE Byte

Type 2 Bit	***LenP*** 2 Byte	***S/C ID*** 8 Byte	***LenE*** 2 Byte

Encrypted Payload LenE Byte

QSNFC: Connection Establishment

- Complete process

QSNFC Connection Establishment

Client		**Server**
	inchoate CH: id_c $\longrightarrow$	
		$(\mathsf{pk}_l, \mathsf{sk}_l) \leftarrow_{\$} \mathsf{KGen}(1^n)$
		$\mathsf{t} \leftarrow \mathsf{Enc}_{\mathsf{sk}_l}(\mathrm{id}_s, \mathrm{time})$
	$\longleftarrow$ RJ: $\mathsf{pk}_l, \mathrm{cert}_s, \mathsf{Sig}(\mathsf{pk}_l), \mathrm{id}_s, \mathsf{t}$	
....	....	....
$(\mathsf{pk}_c, \mathsf{sk}_c) \leftarrow_{\$} \mathsf{KGen}(1^n)$		
$\mathsf{sk}_i \leftarrow (\mathsf{sk}_c, \mathsf{pk}_l)$		
	complete CH: $\mathrm{id}_c, \mathsf{pk}_c, \mathsf{Enc}_{\mathsf{sk}_i}(\mathrm{data}), \mathsf{t}$ $\longrightarrow$	
		$\mathsf{sk}_i \leftarrow (\mathsf{sk}_l, \mathsf{pk}_c)$
		$(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow_{\$} \mathsf{KGen}(1^n)$
	$\longleftarrow$ SH: $\mathrm{id}_s, \mathsf{pk}_s, \mathsf{Enc}_{\mathsf{sk}_i}(\mathrm{data})$	
$\mathsf{sk} \leftarrow (\mathsf{sk}_c, \mathsf{pk}_s)$		$\mathsf{sk} \leftarrow (\mathsf{sk}_s, \mathsf{pk}_c)$

QSNFC: Connection Teardown & Cache Management

- Contrary to TLS that is based on TCP
  - No „connection“ in NFC
  - Actually no teardown is required
- But when is cached information discarded?
  - As soon as there is insufficient memory on the client
- How to decide which information is discarded?
  - We propose to apply cache data replacement strategies
  - Least Frequently Used (LFU), Least Recently Used (LRU), First in, first out (FIFO)
  - Evaluation: no strategy best suited for all scenarios

Outline

1. Introduction & Motivation
2. QSNFC Protocol
    1. System Model & Protocol Stack
    2. Connection Establishment
    3. Connection Teardown & Cache Management
3. Evaluation
    1. Example Use-Cases
    2. Security Analysis
    3. Overhead

Evaluation: Example Use-Cases

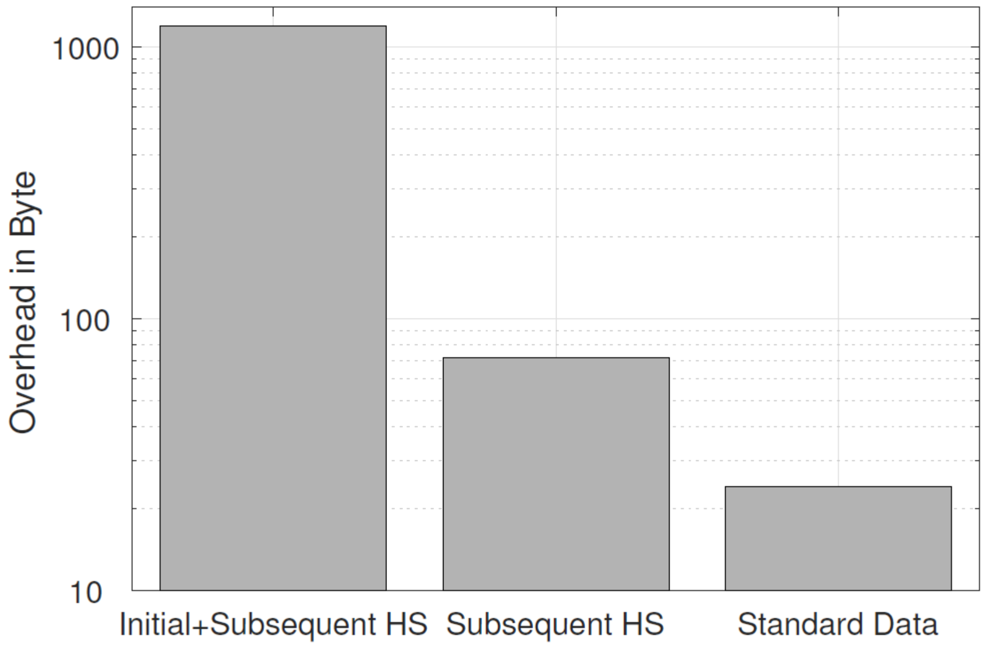
- Card and Reader, e.g. access control system
  - Reader initiates communication → QSNFC client
  - Internet connection for certificate validation
  - More storage for cached information
- Smartphone and IoT device
  - Smartphone initiates communication → QSNFC client
  - Usually, Internet connection available for certificate validation
  - Storage for cached information
- Machine-to-Machine, e.g. Mobile Robot to Machine
  - Role assignment cannot be determined in general
  - Should be chosen such that certificate validation and storage requirements are met

Evaluation: Security Analysis

- Analyize protocol w.r.t. NFC security threats [Haselsteiner & Breitfuß 2006]
- Eavesdropping
  - Confidential information encrypted by AE, only public information unencrypted
- Data Corruption, Data Modification, Data Insertion
  - Detected in confidential data that is protected by AE, unnoticed in unencrypted data → DoS
- Denial-of-Service (DoS)
  - Cannot be mitigated by QSNFC (or any other wireless protocol)
- Man-in-the-Middle
  - Mitigated by certificate based authentication and DH key agreement
- *Physical attacks* (not in [Haselsteiner & Breitfuß 2006])
  - Cannot be mitigated by protocol, but protocol can be implemented on tamper resistant hardware

Evaluation: Overhead

- Evaluated using self-generated certificates: short certificate chain



- Subsequent HS reduces overhead by ~90% compared to initial HS + subsequent HS

Summary

- QSNFC: Secured and efficient protocol for NFC communication
- Uses standard security primitives for easy validation
- Should be suitable for wide range of usage domains
- However, also trade-off must be made
  - For caching, non-volatile memory is required
- The more connection partners that need to be cached:
  - More memory required
  - But: quicker connection establishment with more partners
- QSNFC mitigates most NFC security threats
- Overhead for recurring connections can be reduced by ~90%

Thank you! Any questions?

?

Thomas Ulz

thomas.ulz@tugraz.at

Institute for Technical Informatics
Hardware/Software-Codesign Group
Graz University of Technology