Embedded Systems Specification and Design
Model-based Design and Verification

David Kendall

Introduction

- Security of embedded systems
- Security goals
- Importance of secure protocols
- Modelling and analysis of protocol security

Security of embedded systems: example

Implantable Cardiac Defibrillator (ICD)

- *Pacing*
  Periodically send a small electrical stimulus to the heart
- *Defibrillation*
  Occasionally send a larger electrical charge to restore normal heart rhythm

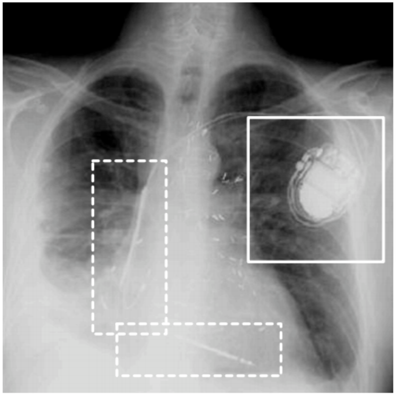


Fig. 1. Chest xray image of an implanted ICD (top right, near shoulder, solid outline) and electrical leads connected to heart chambers (center of rib cage, dotted outline).

ICD programmer

The “programmer” is a device intended to be used to:

- perform diagnostics
- read and write private (patient) data
- adjust therapy settings on the ICD.

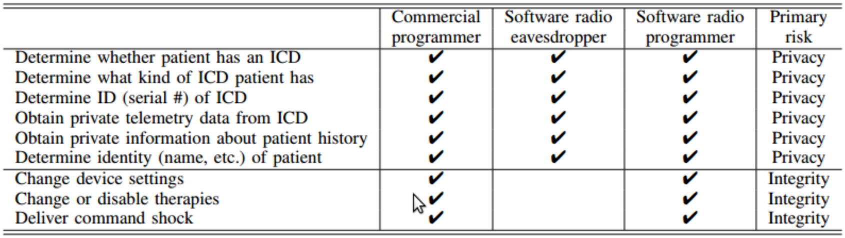
Programmer communicates with ICD wirelessly.

- typically 175 kHz short-range communication

Security of the ICD device

- Halperin, D., Heydt-Benjamin, T.S., Ransford, B., Clark, S.S., Defend, B., Morgan, W., Fu, K., Kohno, T. and Maisel, W.H., *Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses*, IEEE Symposium on Security and Privacy, 129-142, May, 2008
- Considered attacks on ICD security by three classes of attackers:
  - Attacker possessing an ICD programmer
  - Attacker who simply eavesdrops on communications between an ICD and the programmer, using commodity software radio
  - Attacker who eavesdrops as well as generates arbitrary RF traffic to the ICD, possibly spoofing an ICD programmer.
- Demonstrated that successful attacks are possible under all three classes

Experimental results of attacks on ICD



	Commercial programmer	Software radio eavesdropper	Software radio programmer	Primary risk
Determine whether patient has an ICD	✔	✔	✔	Privacy
Determine what kind of ICD patient has	✔	✔	✔	Privacy
Determine ID (serial #) of ICD	✔	✔	✔	Privacy
Obtain private telemetry data from ICD	✔	✔	✔	Privacy
Obtain private information about patient history	✔	✔	✔	Privacy
Determine identity (name, etc.) of patient	✔	✔	✔	Privacy
Change device settings	✔		✔	Integrity
Change or disable therapies	✔		✔	Integrity
Deliver command shock	✔		✔	Integrity

TABLE I
RESULTS OF EXPERIMENTAL ATTACKS. A CHECK MARK INDICATES A SUCCESSFUL IN VITRO ATTACK.

Security goals

- Secrecy
  - Confidential data is not leaked by the system to those not authorised to have it
- Authentication of origin
  - Outputs appearing to come from the system are actually generated by the system
- Integrity
  - System cannot be modified by attacker
- Access and availability
  - System is always able to provide its service (denial of service not possible)

What can we learn from this example?

- Why attacks succeed
  - Messages sent in plaintext
  - No attempt to confirm that messages are received from / sent to an authorised ICD programmer
- Power matters
  - Some attacks simply cause ICD to keep transmitting - depletes its battery
  - Ideally defences should use zero power
- Secure protocols are needed

Importance of secure protocols

- Secure protocols needed to
  - Allow agents to authenticate each other
  - Establish session keys for confidential communication
  - Ensure integrity of data and services
  - Prevent unauthorised access to data and services
- Other components in a secure system include
  - Good cryptographic algorithms
  - Systems security measures for access control
- Security protocols are
  - Often apparently simple
  - Often flawed and vulnerable to unexpected attacks
- Careful design and analysis of security protocols is important
- Following slides are based on, and should be read in conjunction with, chapter 0 of:
  - Ryan, P., Schneider, S., Goldsmith, M. and Lowe, G., *Modelling and analysis of security protocols: the CSP approach*, Addison Wesley, 2000 [local copy]

Notation: Messages

- The steps in a protocol can be represented using the following notation:

$$n. \quad A \rightarrow B : \mathit{data}$$

- This is intended to mean that in the *n*th step of the protocol agent *A* dispatches a message containing *data* to agent *B*.
- N_A is used to represent a *nonce* generated by agent *A*.
- Compound terms are formed by *concatenation* and *encryption*:
  - X, Y denotes the value *X* followed by the value *Y*
  - $\{\mathit{data}\}_K$ denotes the value *data* encrypted using the key *K*
- Example

$$1. \quad A \rightarrow B : \{N_A, A\}_{K_B}$$

  indicates that in the first step of the protocol, agent *A* dispatches a message to agent *B*. The message comprises a nonce generated by *A*, followed by *A*'s name, and is encrypted using *B*'s public key.

Notation: Keys

- Let A and B be communicating agents.
- K_{AB} represents a symmetric key, shared by A and B
- K_A represents an asymmetric public key for A
- $\overline{K}_A$ represents an asymmetric private key for A
- Given K_A it is infeasible to compute $\overline{K}_A$
- $\{\{M\}_{K_A}\}_{\overline{K}_A} = M = \{\{M\}_{\overline{K}_A}\}_{K_A}$

See Chapter 5 in Ross Anderson's book if you need more explanation about symmetric and asymmetric keys, and public key and shared key cryptography.
Anderson, R. *Security Engineering: A guide to building dependable distributed systems* (2nd edition), John Wiley, 2008 [local copy]

Example: Needham-Schroeder Symmetric Key Protocol (NSSK)

- The protocol

$$\begin{array}{ll} 1. & A \to S : \{A, B, N_A\} \\ 2. & S \to A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{SB}}\}_{K_{SA}} \\ 3. & A \to B : \{K_{AB}, A\}_{K_{SB}} \\ 4. & B \to A : \{N_B\}_{K_{AB}} \\ 5. & A \to B : \{N_B - 1\}_{K_{AB}} \end{array}$$

- Notes
  - This is a protocol for authenticated key exchange
  - S represents the key server and K_{SA} and K_{SB} are long term symmetric keys shared between the server and agents A and B, respectively.
  - K_{AB} is the key generated by the key server for agents A and B to use for communication between each other.
  - At the completion of the protocol, A and B share the symmetric key K_{AB} and each knows that the other knows the key!
  - NSSK is the foundation of the Kerberos protocol (which adds timestamps to avoid replay attacks)

Man-in-the-middle attack (outsider)

- Consider this protocol:

$$\begin{array}{ll} 1. & A \rightarrow B : \{X\}_{K_A} \\ 2. & B \rightarrow A : \{\{X\}_{K_A}\}_{K_B} \\ 3. & A \rightarrow B : \{X\}_{K_B} \end{array}$$

- Seems like a neat protocol to allow *A* to send a secret message to *B* without needing to know *B*'s public key
- Relies on the property that $\{\{X\}_{K_A}\}_{K_B} = \{\{X\}_{K_B}\}_{K_A}$
- But can be attacked by an intruder, *I*, who can intercept and insert messages:

$$\begin{array}{ll} 1. & A \rightarrow I(B) : \{X\}_{K_A} \\ 2. & I(B) \rightarrow A : \{\{X\}_{K_A}\}_{K_I} \\ 3. & A \rightarrow I(B) : \{X\}_{K_I} \end{array}$$

- Problem arises because of lack of authentication: *A* does not know who she is talking to.

Needham-Schroeder Public Key (NSPK) Protocol

The protocol

$$\begin{array}{ll} 1. & A \rightarrow B : \{A, N_A\}_{K_B} \\ 2. & B \rightarrow A : \{N_A, N_B\}_{K_A} \\ 3. & A \rightarrow B : \{N_B\}_{K_B} \end{array}$$

- In step 1, Alice sends a message encrypted with Bob's public key, containing her identity and a nonce (one half of the secret)
- In step 2, Bob decrypts the message, extracts Alice's nonce, creates a nonce of his own, and sends a message containing both nonces back to Alice. When Alice receives a message containing her nonce, she assumes the message must have come from Bob (only he could have decrypted her first message) and therefore accepts $\{N_A, N_B\}$ as the shared secret
- In step 3, Alice responds with a message containing Bob's nonce. Bob takes this as confirmation that he is indeed talking to Alice

Attack on NSPK Protocol

Man-in-the-middle attack (insider)

- A and B think this protocol ensures that
  - They have been interacting with each other
  - They agree on the values of N_A and N_B
  - No one else knows N_A and N_B
- But consider this attack by insider I

$$\begin{array}{ll}
1(a). & A \rightarrow I : \{A, N_A\}_{K_I} \\
1(b). & I(A) \rightarrow B : \{A, N_A\}_{K_B} \\
2(a). & B \rightarrow I(A) : \{N_A, N_B\}_{K_A} \\
2(b). & I \rightarrow A : \{N_A, N_B\}_{K_A} \\
3(a). & A \rightarrow I : \{N_B\}_{K_I} \\
3(b). & I(A) \rightarrow B : \{N_B\}_{K_B}
\end{array}$$

- Here the intruder, I, is a legitimate user of the network (insider) but behaves dishonestly by masquerading as Alice, $I(A)$, when communicating with Bob

Man-in-the-middle attack (insider)

- This very famous attack was discovered by Gavin Lowe, using a model-checker, and published in 1995, 17 years after the original publication of the protocol. See Lowe, G., *An Attack on the Nedham-Schroeder Public-Key Authentication Protocol*, Information Processing Letters, 56:3, pp. 131–133, 1995 [local copy]
- It relies on a dishonest insider, *I*, interleaving two runs of the protocol.
- At the end of the attack, *A* believes that she shares knowledge of N_A and N_B exclusively with *I*. While *B* believes that he shares knowledge of N_A and N_B exclusively with *A*. Neither belief is correct.
- Notice that the attack does not involve breaking any cryptography.
- It is possible just by the nature of the protocol.

Modelling and analysis of protocol security

- Lowe's use of a model-checker to discover the attack on NSPK protocol stimulated much research in this area.
- Now many protocols have been analysed with general-purpose and custom-built model-checkers.
- Dedicated tools for model-checking security include
  - AVISPA - applied to a wide range of security protocols (see the AVISPA home page)
  - ProVerif - applied to TLS and InfoCard
  - Scyther - applied to MQV family, ISO/IEC 9798 standard, IKE key exchange protocols in IPSEC
- General-purpose model checking of security
  - TLA$^+$ (e.g. Automatic vulnerability checking of IEEE 802.16 WiMAX protocols through TLA$^+$)
  - UPPAAL (e.g. Timed model-checking of security protocols)
  - SPIN (e.g. Generic and efficient attacker models in SPIN)
- For an excellent overview see Model Checking Security Protocols

Modelling and analysis of protocol security

The Dolev-Yao assumptions, Dolev, D., Yao, A., On the security of public key protocols, IEEE Trans. on Information Theory, 29:2, March 1983

A perfect public key system

- the one-way cryptographic functions are unbreakable, e.g. the only way to decrypt a message is by having possession of the correct key
- public directory is secure and tamper-proof
- everyone has access to the public keys; only their owners know the private keys

A powerful intruder

- Full access to the network
- Considered to be a legitimate user of the network: can properly initiate and receive messages
- Can intercept, forward, duplicate, drop, or replay any message
- Can apply standard functions to any known data to create and send new messages

Modelling and analysis of protocol security

Example: Finding the flaw in the NSPK protocol using UPPAAL

Data and function declarations

```
urgent chan network;  // A single urgent channel models the whole network

typedef struct {
          int[0,15] key;
          int[0,15] data1;
          int[0,15] data2;
} encrypted_t;       // data1 and data2 encrypted with the key

typedef struct {
          int[0,15] mtype;     // Message type : msg1, msg2 or msg3
          int[0,15] receiver;  // Intended receiver
          encrypted_t crypt;   // Encrypted part of the message
} message_t;

const int[0,15] A = 0;         // Alice
const int[0,15] B = 1;         // Bob
const int[0,15] I = 2;         // Intruder
const int[0,15] msg1 = 3;
const int[0,15] msg2 = 4;
const int[0,15] msg3 = 5;
const int[0,15] nonce[3] = {6, 7, 8};  // Nonces for {Alice, Bob, Intruder}
const int[0,15] key[3] = {9, 10, 11};  // Keys for {Alice, Bob, Intruder}
const int[0,15] null = 12;             // Dummy data

int[0,15] partner[3];          // Partners for {Alice, Bob, Intruder}

meta message_t tmp;            // Auxiliary variable for message-passing

// Constructors for encrypted_t and message_t
encrypted_t E(int[0,15] key, int[0,15] data1, int[0,15] data2) {
  encrypted_t result = {key, data1, data2};
  return result;
}

message_t M(int[0,15] mtype, int[0,15] receiver, int[0,15] key, int[0,15] data1, int[0,15] data2) {
  encrypted_t e = E(key, data1, data2);
  message_t result = {mtype, receiver, e};
  return result;
}

bool isAgent(int[0,15] v) {
  return ((v == 0) || (v == 1) || (v == 2));
}
```

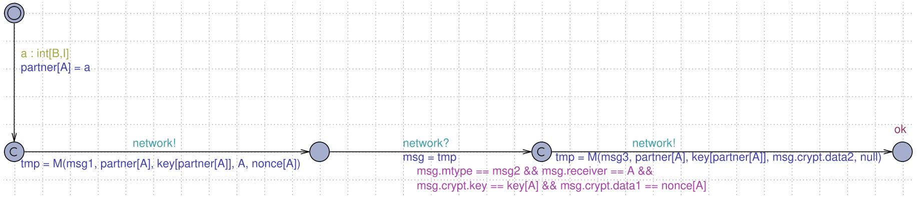
Modelling data and functions in a security protocol

General principles

- Abstract from the details of all cryptographic functions and data
- There's no need to actually encrypt and decrypt messages – just to indicate which parts of a message are encrypted and with what key
- The keys just need to be distinguishable from other data in the model
- Similarly, there's no need to actually create nonces cryptographically – just to be able to distinguish each nonce from other data in the model
- Simplify, simplify, simplify!!!

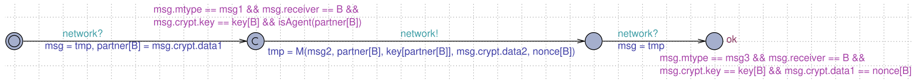
The Alice and Bob models for NSPK

Alice



a : int[B,I]
partner[A] = a
network!
tmp = M(msg1, partner[A], key[partner[A]], A, nonce[A])
network?
msg = tmp
msg.mtype == msg2 && msg.receiver == A &&
msg.crypt.key == key[A] && msg.crypt.data1 == nonce[A]
network!
tmp = M(msg3, partner[A], key[partner[A]], msg.crypt.data2, null)
ok
C
C

Bob

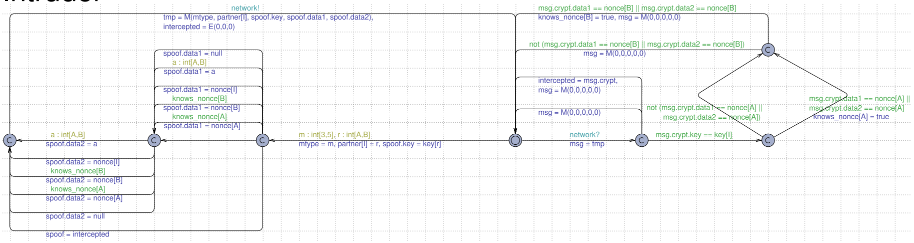


network?
msg = tmp, partner[B] = msg.crypt.data1
msg.mtype == msg1 && msg.receiver == B &&
msg.crypt.key == key[B] && isAgent(partner[B])
network!
tmp = M(msg2, partner[B], key[partner[B]], msg.crypt.data2, nonce[B])
network?
msg = tmp
msg.mtype == msg3 && msg.receiver == B &&
msg.crypt.key == key[B] && msg.crypt.data1 == nonce[B]
ok
C

- Honest agents, Alice and Bob, engage in a single run of the protocol

The Intruder Model for NSPK

Intruder



- Highly non-deterministic model of dishonest insider, implementing full Dolev-Yao capabilities
- Goal: to allow model-checker to discover possible attacks

Notes on the agent models

- Alice and Bob are honest agents
- The models of Alice and Bob show their participation in a single run (session) of the protocol, following the protocol specification
- The Intruder is a dishonest agent with insider privileges
- The Intruder model is highly non-deterministic, giving the Intruder all the capabilities of a Dolev-Yao attacker
- The Intruder model is *not* a model of an implementation of a possible intruder
  - an implementation of this Intruder acting on a physical network would be discovered quickly
- The Intruder model is designed to represent all possible intruder actions so that a model-checker can discover which combinations of actions represent possible attacks
- If an attack is discovered, a more subtle intruder can be implemented

Properties of the NSPK protocol model

There's some execution where the protocol works as intended (**TRUE**)

```
E<> (Alice.ok && Bob.ok && partner[A] == B && partner[B] == A
     && !(Intruder.knows_nonce[A] || Intruder.knows_nonce[B]))
```

If Alice and Bob have completed their runs, then, if Alice's partner is Bob, Bob's partner is Alice and the intruder knows neither of the nonces (**TRUE**)

```
A[] ((Alice.ok && Bob.ok) imply ((partner[A] == B) imply
      (partner[B] == A && !(Intruder.knows_nonce[A] ||
                             Intruder.knows_nonce[B]))))
```

If Alice and Bob have completed their runs, then, if Bobs's partner is Alice, Alice's partner is Bob and the intruder knows neither of the nonces (FALSE)

```
A[] ((Alice.ok && Bob.ok) imply ((partner[B] == A) imply
      (partner[A] == B && !(Intruder.knows_nonce[A] ||
                             Intruder.knows_nonce[B]))))
```

And finally, . . .

- Demo of the use of UPPAAL to check properties of the NSPK protocol
- Consideration of the implications of the attack on the NSPK protocol
- Fixing the NSPK protocol
- Summary

References

- Koopman, P., *Embedded system security*, IEEE Computer, 37(7): 95-97, July 2004 [pdf]
- Halperin, D., Heydt-Benjamin, T.S., Ransford, B., Clark, S.S., Defend, B., Morgan, W., Fu, K., Kohno, T. and Maisel, W.H., *Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses*, IEEE Symposium on Security and Privacy, 129-142, May, 2008 [pdf]
- Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, and Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage, *Experimental Security Analysis of a Modern Automobile*, IEEE Symposium on Security and Privacy 2010, 447-462, 2010 [pdf]
- Basin, D., Cremers, C., and Meadows, C., *Model Checking Security Protocols*, in Handbook of Model Checking, eds. Clarke, E., Henzinger, T., Veith, H., and Bloem, R., Springer International Publishing, 2018