Chapter 1
Emerging Security Challenges for Ubiquitous Devices

Mirosław Kutyłowski, Piotr Syga, and Moti Yung

Abstract In this chapter we focus on two important security challenges that naturally emerge for large scale systems composed of cheap devices implementing only symmetric cryptographic algorithms. First, we consider threats due to poor or malicious implementations of protocols, which enable data to be leaked from the devices to an adversary. We present solutions based on a watchdog concept—a man-in-the-middle device that does not know the secrets of the communicating parties, but aims to destroy covert channels leaking secret information. Second, we deal with the problem of tracing devices by means of information exchanged while establishing a communication session. As solutions such as Diffie-Hellman key exchange are unavailable for such devices, implicit identity information might be transmitted in clear and thereby provide a perfect means for privacy violations. We show how to reduce such risks without retreating to asymmetric algorithms.

1.1 Introduction

The popularity and wide spread of ubiquitous systems requires us to focus our attention on possible threats and challenges that are either not present or are easy to solve in other environments. Quite often, a user of such a system is in possession of multiple severely constrained devices that may communicate with others without the user's explicit consent or knowledge. Since the user has no direct control over the messages that are exchanged, a malicious manufacturer may aim to leak users' secrets over a covert channel created when random values should be transmitted. The problem is acute, since due to cost factors it is hard to defend against it—e.g., by going through a tough certification process.

M. Kutyłowski · P. Syga
Wrocław University of Science and Technology, Wrocław, Poland

M. Yung (✉)
Columbia University, New York, NY, USA
e-mail: moti@cs.columbia.edu

G. Avoine, J. Hernandez-Castro (eds.), *Security of Ubiquitous Computing Systems*,
https://doi.org/10.1007/978-3-030-10591-4_1

Another threat which is specific to ubiquitous systems is the possibility of tracking a device even if it communicates over encrypted channels: e.g., establishing a shared key by two devices may enable identifying these devices. Device tracking may result in various personal threats, ranging from profiling the device holder (resulting in targeted advertisements) to criminal activities such as stalking and impersonation. Apart from risks to the individuals, there is a threat of massive tracking being done for illegal business purposes, organized crime, or subversive or terrorist purposes, as well suppressing personal freedom. So far, methods for preventing tracking have not been adequately developed. In this chapter we identify the aforementioned threats and present some solutions that are feasible in ubiquitous systems.

The chapter is organized into two main parts. In Sect. 1.2 we focus on the threat posed by a device designed to leak the user's secrets via covert channels. As a countermeasure against leaking information in supposedly random parts of a communication, we propose using a watchdog device. We describe modifications of cryptographic primitives that allow the use of a watchdog: in Sects. 1.2.3.1 and 1.2.3.2 we discuss commitments, in Sects. 1.2.3.3 and 1.2.3.4 we focus on generating and answering challenges, and Sect. 1.2.3.5 is devoted to distance bounding. In Sect. 1.3 a privacy aspect of the protocols is discussed, in particular we aim to prevent the adversary from identifying a device as partaking in two different communications. Section 1.3.2 discusses some basic ideas for coping with communication linking in ubiquitous systems using predistributed keys. In Sect. 1.3.3 we present a notion of using 'general pseudonyms', common to a group of devices, in order to preserve their privacy, whereas in Sects. 1.3.4 and 1.3.5 we discuss modifying the transmitted keys by a permanent evolution in the former case, and by transmitting the identifiers with a pseudorandom perturbation in the latter.

1.2 Malicious Devices and Watchdog Concept

1.2.1 Attacks by Malicious Devices

In ubiquitous systems, controlling hardware devices in terms of their security features may be a major problem. Methods such as certification of products, strict manufacturing regimes and controlling the supply chain might be relatively costly compared with the purpose and application area of such systems. On the other hand, there are numerous smart methods to cheat users: a device may look inoffensive, while it may leak secret data to an adversary using various types of covert channels. In fact, no extra message has to be sent: covert channels can be hidden in regular innocent-looking messages. The main problem areas are:

1. Protocol steps where random elements are created by a malicious device and subsequently presented in some form.
2. Erroneous executions where (random) faults in protocol execution may encode information.

Note that in the second case there might be no security alert. For instance, distance bounding protocols explicitly admit failures.

While the most effective methods of the types mentioned in point 1 require asymmetric cryptography (see, e.g., [590]), there are possibilities to create narrow covert channels even if a random string is processed with, say, a hash function. Assume for instance that the device chooses r at random, while $\mathrm{Hash}(r)$ is available to the adversary. Assume that a malicious device executing the protocol intends to leak bits of a string μ of length 2^l. Assume also that it has a secret key k shared with the adversary, who is observing the communication channel. Instead of just choosing r at random, computing $\mathrm{Hash}(r)$, and presenting it, the following steps are executed:

1. choose r at random, $\quad s := \mathrm{Hash}(r), \quad z := \mathrm{Hash}(s, k)$,
2. parse z as $a||b||\ldots$, where a consists of l bits and b consists of m bits,
3. goto 1, if the m-bit substring of μ starting at position a is different from b,
4. output s.

Of course, this is a *narrow channel*, as m must be small enough so that an appropriate r can be found in a reasonable time—in practice we are talking about a few bits per protocol execution. Note that this procedure works for any hash function. It works also if the loop may be executed only a limited number of times and, during the last loop execution, steps 2 and 3 are omitted. Unfortunately, if a device is delivered as a black-box, then the possibilities to inspect what the device is doing are severely limited. There are only a few exceptions (see, e.g., [94]).

1.2.2 Active Watchdog Concept

The general idea is that a device is controlled by its dedicated watchdog coming from a source independent from the device manufacturer and the device provider. The watchdog should detect malicious operation or make it ineffective, given that no access to the internal operation of the device controlled is granted. A watchdog is an active device, modeled as follows:

- it controls the whole communication between the device and the external world,
- it can delete and change all messages,
- it may challenge the device during extra interactions executed by them.

The idea of separating the tasks between two independent devices is an old concept [136]. The same concerns supervising computation correctness by an external unit [97]. Nevertheless, it has attracted more attention in the post-Snowden era, being used, among others, in a solution guarding against subversion attacks on cryptographic devices—see the paper [31] and an extension [510].

1.2.3 Solution Strategy

We propose the following general approach for transforming cryptographic protocols into versions involving a watchdog:

- the proposed changes in a protocol should be minimal, preferably exactly the same protocol should be executed by other protocol participants,
- we identify the basic components of the protocol that enable creating covert channels and for each of them provide a modified secure version.

The main problem areas are the steps that are either nondeterministic or not verifiable by the protocol partners and external observers. This concerns in particular choosing elements at random. However, we have also to consider deterministic steps if their correctness can be verified only with a delay—note that in the meantime the session can be interrupted due to, for example, a real or claimed physical fault.

From now on we use the following terminology:

Device	a potentially malicious device to be controlled,
Watchdog	a watchdog unit controlling the Device,
Reader	the original partner in the protocol executed by the Device.

Apart from that, we talk about an adversary that may observe and manipulate communications between the Watchdog and the Reader, while the adversary has no access to communications between the Watchdog and the Device (including, in particular, all Device's output.)

1.2.3.1 Commitments: Problems with Solutions Based on Hash Functions

Cryptographic hash functions are frequently used to generate commitments in lightweight protocols for ubiquitous devices. Due to their one-wayness, it is hard to build a broad subliminal channel, however the following attacks are still possible:

- choosing the committed value in a malicious way (e.g., from a small subspace),
- creating a narrow covert channel according to the method described on page 5.

For this reason, forwarding a hash value by the Watchdog should not occur unless the hash argument has been randomized by the Watchdog. One may attempt to randomize a hash value in the following naïve way:

1. the Device chooses r' at random, computes $c' := \mathrm{Hash}(r')$ and sends c' to the Watchdog,
2. the Watchdog selects ρ at random and returns it to the Device,
3. the Device computes the final value $r := r' \oplus \rho$ (where $\oplus$ stands for the bitwise XOR operation) and sends the final commitment $c := \mathrm{Hash}(r)$.

The problem with this approach is that in the case of standard hash functions, the Watchdog cannot check that c has been computed correctly without retreating to very complicated procedures (more expensive than simple commitments based on

symmetric cryptography) or disclosing r. However, revealing the committed value must not occur before the moment when this value is transmitted to the Reader. Indeed, the Watchdog might transfer this value prematurely—we cannot exclude the possibility that the Watchdog and the Reader are colluding.

Recall that randomization of the hash argument is easy for Pedersen commitments:

- the Device chooses r' and s' at random and computes $c' := g^{r'} \cdot h^{s'}$, it presents c' to the Watchdog,
- the Watchdog chooses r'', s'' at random computes $c := c' \cdot g^{r''} \cdot h^{s''}$ and:
  - sends r'', s'' to the Device,
  - sends the commitment c to the Reader,
- the Device computes the committed values: $r := r' \cdot r''$, $s := s' \cdot s''$.

Unfortunately, such commitments require implementing asymmetric cryptography, while we have assumed that we are limited to symmetric methods. Concluding, as it seems very hard to overcome the problems related to hash functions that are not based on asymmetric cryptography, we must focus on symmetric encryption. Note that symmetric encryption has to be implemented on most ubiquitous devices to encrypt messages, so reusing it for commitments may reduce the implementation cost and simplify the hardware requirements.

1.2.3.2 Commitments Based on Symmetric Encryption

We assume that the encryption function works with n-bit keys and converts n-bit blocks into n-bit ciphertexts. Hence each key defines a permutation on n-bit blocks. We assume that the encryption scheme is resistant to known-plaintext attacks.

The second assumption is that given a ciphertext c, for most plaintexts t there is a key k such that $c = \mathrm{Enc}_k(t)$.

Basic Commitment Mechanism

1. choose a plaintext t and a key k at random,
2. compute $c := \mathrm{Enc}_k(t)$,
3. present (t, c) as a commitment for k.

In order to open the commitment (t, c) one has to present k. The commitment opening test is $\mathrm{Enc}_k(t) \stackrel{?}{=} c$. Note that breaking the commitment is equivalent to a successful known-plaintext attack. Of course some care is required when choosing the encryption scheme, as each single bit of the key has to be secure against cryptanalysis.

Controlled Commitment Mechanism

1. The Device creates a commitment (t', c') for k' using the basic mechanism (i.e., $c' := \mathrm{Enc}_{k'}(t')$ and t' is a single n-bit block), and presents (t', c') to the Watchdog,

2. The Watchdog chooses n-bit blocks θ and κ at random and presents them to the Device,
3. The Device recomputes the basic commitment:

$$t := t' \oplus \theta, \quad k := k' \oplus \kappa, \quad c := \mathrm{Enc}_k(t)$$

and presents c to the Watchdog,
4. the Watchdog chooses α at random and computes

$$t := t' \oplus \theta, \quad \zeta := \mathrm{Enc}_c(\alpha)$$

5. The Watchdog presents the final commitment (t, α, ζ) concerning the element k to the Reader.

Note that the Watchdog has no control over the correctness of execution of the third step and therefore the Device can potentially install a covert channel in c. However, at this moment c is hidden behind the commitment (α, ζ) chosen by the Watchdog. So in particular, the potential covert channel is hidden as well.

However, it is necessary to note that the Device should not be given the freedom to interrupt the protocol execution. Otherwise the Device could selectively interrupt the execution depending on the values of t and k and in this way leak some secret bits.

Opening the commitment is controlled by the Watchdog in the following way:

1. the Device presents k' to the Watchdog,
2. the Watchdog computes $k := k' \oplus \kappa$ and aborts if

$$c' \neq \mathrm{Enc}_{k'}(t'), \quad \text{or} \quad c \neq \mathrm{Enc}_k(t' \oplus \theta),$$

3. the Watchdog presents k to the Reader,
4. the Reader computes $\bar{c} := \mathrm{Enc}_k(t)$, and accepts the opening if $\zeta = \mathrm{Enc}_{\bar{c}}(\alpha)$.

Note that breaking this commitment by the adversary is at least as hard as known-plaintext cryptanalysis. Namely, for a (plaintext, ciphertext) pair (α, ζ) one can choose t at random and present it to the adversary. Indeed, for any key ψ that is a solution for (α, ζ), for most values t there is a key k such that $\psi = \mathrm{Enc}_k(t)$.

1.2.3.3 Encrypted Random Challenge

We assume that the protocol requires the Device to send a random challenge r as a ciphertext $\mathrm{Enc}_k(r)$ for the key k shared with the Reader. The following protocol enables the Watchdog to control the situation:

1. the Watchdog sends a commitment θ to an element α chosen at random (for this purpose the basic commitment from Sect. 1.2.3.2 can be used),

2. the Device chooses r' at random, computes $s := \mathrm{Enc}_k(r')$ and sends s to the Watchdog,
3. the Watchdog computes $\sigma := \alpha \oplus s$ and sends
   - σ to the Reader,
   - α to the Device (if necessary, the Watchdog attaches also an opening to the commitment).

With this approach, the random challenge is $r = \mathrm{Dec}_k(\sigma)$.

For the Device, the final shape of r is unpredictable so there is no way to hide information in r. On the other hand, the Watchdog cannot influence r (for instance, enforce repetition of the same r), as α has to be determined before the (random) ciphertext s is presented to it.

Note that if the random challenge is presented in clear, then a simplified version of the above procedure can be used.

1.2.3.4 Answers to Challenges

One of the moments when information can be leaked to the adversary is when the Device is responding to a challenge sent by the Reader by computing some deterministic algorithm, but any verification procedure for the response requires knowledge of a key shared by the Reader and the Device. This key must not be available to the Watchdog due to security reasons. In this case no censorship by the Watchdog is possible. On the other hand, the Reader may discover the manipulation when it is already too late, since the malicious message has been already on the air, available to the adversary.

The solution to this problem is to encrypt the communications from the Watchdog to the Reader with a random session key that is unknown to the Device. Such an encryption results in randomization destroying, any covert channel.

The procedure to establish the session key is as follows:

1. the Reader creates a commitment u to a random key z' and sends u to the Watchdog of the Device,
2. at the same time the Reader creates a ciphertext $c_0 = \mathrm{Enc}_k(z')$ with the key k shared with the Device, and sends it to the Device through the Watchdog,
3. the Device decrypts c_0 and reveals z' to its Watchdog,
4. the Watchdog checks the correctness of z' against the commitment u and chooses a key z at random,
5. the Watchdog sends $\mathrm{Enc}_{z'}(z)$ to the Reader,
6. from now on, in the current session all messages forwarded by the Watchdog from the Device to the Reader are additionally encrypted with the key z.

Note that for the steps 1 and 2 one can apply the procedures from Sects. 1.2.3.2 and 1.2.3.3. Thereby, the protocol can be secured against a malicious Reader as well. Note also that the Device knows z' so it could learn z from $\mathrm{Enc}_{z'}(z)$. However, the

Device cannot access the communication channel without help from the Watchdog. Consequently, it does not know $\mathrm{Enc}_{z'}(z)$.

For the adversary, the situation is just the opposite: it has access to $\mathrm{Enc}_{z'}(z)$, but it does not know z' and cannot learn it from the colluding Device, as there is no communication channel between them.

1.2.3.5 Distance Bounding Protocols

A distance bounding protocol [112] is executed when the physical presence of the Device in a close proximity of the Reader needs to be verified. This is particularly important in scenarios concerning access control in the presence of hardware tokens. By utilizing the timing delays between sending out a challenge bit and receiving a response, the Reader can calculate an upper bound on the distance to the verified device, and, if the Device is outside the intended perimeter, abandon the protocol.

The main and most problematic part in distance bounding protocols is the Rapid Bit Exchange (RBE) phase that is key to calculating the distance based on the response time. Typically, it is executed by a Device and a Reader as follows:

1. the Device and the Reader share m-bit strings r_0 and r_1 (computed from a shared secret and a nonce transmitted in clear),
2. for $i = 1, \ldots, m$, the following steps are executed:
   (a) the Reader sends a pseudorandom bit $c[i]$,
   (b) the Device responds immediately with $r_{c[i]}[i]$.
3. The Reader aborts the protocol if the answers of the Device have not been obtained within (strict) time limits or not all of them are correct.

The RBE phase is potentially a source of problems:

- The Device can send bits different from $r_{c[i]}[i]$ just to create a covert channel. Incorrect responses might be interpreted not as malicious activity of the Device, but as an attempt by an adversary standing between the Device and the Reader to cheat about the distance.
- Convincing the Watchdog about the correctness of the responses is a challenge: during the RBE phase there is no time for exchange of messages between the Device and the Watchdog and for nontrivial computations.

The solution presented below is based on blinding the responses with a one-time pad, where the random key is transported via the Device to the Watchdog in a secure way. It consists of the three phases described below:

PREPARATION: During this phase a blinding string is prepared. Namely, a session key z shared between the Watchdog and the Reader is established, as described in Sect. 1.2.3.4. However, instead of using it for encapsulation of messages it is used as a seed for creating a pseudorandom bit string $B(z)$.

RBE: RBE is executed as in the standard protocol except that the Watchdog, after receiving the ith response $A[i]$ from the Device, converts it to

$$A'[i] := A[i] \oplus B(z)[i]$$

and forwards $A'[i]$ to the Reader.

VERIFICATION: Aside from the usual operations, the Reader additionally removes the blinding by XORing each $A'[i]$ with $B(z)[i]$.

1.3 Privacy

Deploying ubiquitous systems means not only substantial advantages for many application areas, but at the same time emerging and significant privacy problems.

For instance, if device identifiers are explicitly transmitted when two devices establish a communication session, then it is relatively easy to create a global tracking system and misuse the information collected in this way. Note that in the case of symmetric cryptography, one cannot first securely establish a session key (e.g., with the Diffie-Hellman protocol), and then send the identifiers encrypted with this key. Theoretically, parties A and B sharing a symmetric key k may recognize each other without transmitting any identifier:

1. party A chooses a nonce n at random and for each of its partners i:
   (a) chooses a nonce n_i at random,
   (b) computes $M_i = \mathrm{Enc}_{k_i}(n, n_i)$, where k_i is the key shared with this partner,
2. party A broadcasts $(n, M_1, M_2, \ldots,)$,
3. party B decrypts each M_i with all shared keys it knows; if the first half of the plaintext is n, then this is the right key and it responds with n_i.

The obvious problem with such a protocol is its lack of scalability, so it cannot be applied in large-scale ubiquitous systems.

Fortunately, in the case of ubiquitous systems the size of the system may be an advantage: even if the adversary can monitor communication at many places, one can assume that the adversary is not omnipresent and consequently only a fraction of the interactions are available to him. Some defense methods are based on this fact.

Another factor that may hinder adversarial actions is that a device may have some a priori knowledge about its potential communication partners and therefore its computation effort can be limited to these partners. On the other hand, an adversary having no idea who is attempting to communicate may be forced to consider all possible pairs of partner devices—the computational effort in this case might be higher by an order of magnitude. Moreover, it may lead to many false candidates—while for the attacked device this problem does not arise, as its range of partner choices is limited.

The rest of this section is devoted to diverse methods that at least reduce the privacy risks without retreating to asymmetric cryptography.

1.3.1 Symmetric Protocols and Deniability

From the point of view of privacy protection, symmetric cryptography leads to fewer problems than advanced methods based on asymmetric cryptography. Namely, if all cryptographic secrets used to execute a protocol are available to both parties executing the protocol, then a valid transcript of a protocol execution can be created by either party of the protocol. Therefore such a transcript cannot be used as a proof that an interaction has taken place.

Let us note that the situation might be different if a secret for symmetric cryptography is known only to one party of the protocol. For instance, if a device D is authenticated by presenting a token s, where $\mathrm{Hash}(s)$ is a one-time key publicly known as belonging to D, then presenting s may serve as a proof that an interaction with D has taken place.

1.3.2 Identity Hiding with Random Key Predistribution

When tools such as Diffie-Hellman key exchange are unavailable, random key predistribution may help to protect the initial information exchange. According to this approach, a session is initiated as follows:

Phase 1 (key discovery): the communicating devices find keys from the key predistribution scheme that they share,

Phase 2 (identity disclosure): the devices exchange their identity information, and communication is protected with the shared keys found in Phase 1,

Phase 3 (authentication and key establishment): the devices continue in a way tailored to the declared identity information, encryption may be based on bilateral keys and not on the keys from the key predistribution.

Let us recall a few details of key predistribution schemes. There is a global key pool $\mathscr{K}$ of size N. We assume that each device i has an individual key pool $\mathscr{K}^{(i)}$ consisting of keys $k_1^{(i)}, \ldots, k_n^{(i)}$. If devices i and j meet, they identify a key (or keys) in $\mathscr{K}^{(i)} \cap \mathscr{K}^{(j)}$ (shared keys).

There are certain details about how to choose the subsets of keys to ensure that the set of keys $\mathscr{K}^{(i)} \cap \mathscr{K}^{(j)}$ is nonempty. The basic approach is to choose random subsets—then due to the birthday paradox there is a fair chance of a key being shared if $n \approx \sqrt{N}$. Another approach is to consider a projective space and assign keys corresponding to a line to each device—note that in a projective space every two lines intersect [124].

Key predistribution may reduce the privacy risks during identity information exchange as most of the devices in the vicinity of devices A and B initializing an interaction will not be able to access the information exchanged in Phase 2. However, some details have to be implemented carefully:

- In Phase 1 a device cannot simply send identifiers of the keys it holds as this set of identifiers would serve as its implicit identifier and can be abused to trace it.
- An adversary that knows a key k from the pool (e.g., as a legitimate user) would be able to trace all interactions in Phase 2 between devices for which k is the shared key. Even worse, the adversary may attempt to learn as many keys from the pool as possible, e.g., by hacking its own devices.

In the following we recall a few techniques that reduce these risks.

1.3.2.1 Key Discovery with a Bloom Filter

Bloom filters can be used as a compact data structure to enable discovery of shared keys in a relatively secure way. A Bloom filter is a bit array of length, say, 2^l. In order to "insert" keys $k_1, \ldots, k_t$ into a filter a device A performs the following steps:

1. initialize a Bloom filter F as an array of zeroes,
2. choose a nonce η at random,
3. for each $i \leq t$ "insert" the key k_i into the filter:
   (a) for $j \leq m$ compute $\text{Hash}(\eta, k_i, j)$, truncate it to the l most significant bits, getting $h_{i,j}$ (m is a Bloom filter parameter),
   (b) set the bits of F to 1 in the positions $h_{i,1}, \ldots, h_{i,m}$,

When a device B receives the filter F together with the nonce η, for each key it holds it can perform a similar calculation and check whether F holds only ones in the computed positions. If there is even a single 0, this key is not shared with A. Otherwise, it is a candidate shared key. For some details see [332].

Of course, depending on the size of the Bloom filter, the number of keys inserted into the filter and the parameter m, there might be false candidates. In order to inform A about the candidate keys one can reply with a Bloom filter created for the candidate keys. A few interactions of this type should suffice to narrow the set of candidates to the set of shared keys on both sides.

1.3.2.2 Multiple Shared Keys

For the sake of privacy preservation, it is useful to design the key predistribution scheme so that two devices share multiple keys. Then during Phase 2 the devices sharing keys, say, $k_{i_1}, \ldots, k_{i,w}$, encrypt each message with all these keys. For instance, one can use single encryption with a key $K := \text{KGF}(k_{i_1}, \ldots, k_{i,w}, \textit{nonce})$ where KGF is a key generation function, and *nonce* is a nonce exchanged in the clear.

The advantage of this approach is that the adversarial device needs to know all keys $k_{i_1}, \ldots, k_{i,w}$ to decrypt the communications of Phase 2. Of course, some keys from the list may be known to the adversary—due to the key predistribution mechanism.

1.3.2.3 Epoch Keys

Paradoxically, reducing the number of keys in a pool may be advantageous. According to [146], each device holds the following:

long term keys: These are the keys from a key predistribution scheme. Long term keys are used only to decrypt ciphertexts containing epoch keys,

epoch keys: These keys are used for establishing communication links within their epoch as described in previous subsections.

From time to time the system provider runs the following steps:

1. new epoch keys are generated at random,
2. for each key k from the pool of long term keys, the corresponding ciphertext $C := \mathrm{Enc}_k(\eta)$ with a MAC is created, where η is an epoch key selected for k,
3. the ciphertexts of the epoch keys are disseminated to the devices.

There might be different ways of dissemination. For example, it might be done by broadcasting over a public radio channel or handling the new epoch keys when a device logs into the system.

The crucial property of this approach is that the number of epoch keys is N/m, where N is the number of long term keys, and that an epoch key is assigned to m long term keys when creating the ciphertexts described above. The parameter m could be a small constant such as 4. If each device holds $n \approx \sqrt{N}$ long term keys, and m is a small constant, then the expected number of epoch keys shared by two devices is approximately m.

Using epoch keys has advantages from the point of view of privacy protection:

- As the number of shared keys increases the probability that a different device can follow the communication in Phase 2 is significantly reduced. For instance, for $m = 2$ the probability changes from n/N to

$$\frac{\binom{n}{2}}{\binom{N/2}{2}} \approx \frac{n^2/2}{N^2/8} = 4 \cdot \left(\frac{n}{N}\right)^2 .$$

  As typically $\frac{n}{N} \ll 1$, the progress is significant.
- Devices A and B may share epoch keys for two reasons:
  - A and B share a long term key, so consequently they share the epoch key encrypted with this long term key,
  - the same epoch key has been encrypted with different long term keys possessed, respectively, by A and B.

In the second case the same pair of long term keys does not necessarily correspond to the same epoch key during the next epoch. So an adversary that may follow A and B in one phase cannot automatically continue to do so in the next epoch.

1.3.3 Overloading Identifiers

The idea is to follow the approach used in human society: in everyday life we do not use unique identifiers (like personal identification numbers), but ambiguous identifiers such as first names (in Europe), family names (in China), etc. Despite the fact that the same identifier is used by a large number of people, there is almost no confusion in social interactions. For ubiquitous systems, we can mimic this approach. Namely, we assume that:

- The number of devices is large, say N, however in each local environment the number of devices is at most m, where $m \ll N$,
- Apart from its main unique identifier, each device holds k identifiers from a pool of size M, where $M \ll N$ and k is a small constant,

Now, the concept is that in a local environment each device occurs under one of its short IDs [147]. The process of joining such an environment involves using one of its short identities not yet in use there so far. The chances of having such an identifier available might be surprisingly high due to the famous power-of-two-choices phenomenon [46]. The crucial point in this approach is to determine how many short identifiers are needed globally, as minimizing their number provides better privacy protection. One can show that this process proceeds successfully for m participants (i.e., they always have the ability to choose an unused identifier) with probability at least p if the number of short identifiers N is approximately

$$\left(-\frac{m^{k+1}}{(k+1)\ln(1-p)}\right)^{1/k}.$$

1.3.4 Pairwise Keys Evolution

If a device has no asymmetric cryptography implemented, then establishing a bilateral key with another device is a problem. In small scale systems, we can deploy such keys in a secure environment during the manufacturing phase. Unfortunately, this is no longer possible in most large-scale application scenarios. So, the common solution is to pair two devices in a private environment and hope that electronic communication has not been tapped there. Only in a limited number of cases the key (or some auxiliary information) can be carried by the user and inserted into both devices. Indeed, a device may have no appropriate interface for such direct

communication with the user (e.g., a keyboard). Another issue is that users might feel annoyed by the need for such manual work.

A pragmatic solution to this problem was presented in [499]:

- One can assume that the adversary is not omnipresent, hence it misses some number of successful communications between each pair of devices.
- Each time two devices meet, they change the bilateral key in a random way, i.e., if the shared key is k, then new key is $F(k, i)$, where $i \leq n$ is chosen at random and $F(k, i)$ means k with its ith bit flipped.

This approach has the advantage that the bilateral key evolves in an unpredictable way and, after a small number of successful interactions it, becomes uniformly distributed in the key space. If during this time the adversary is not monitoring communication, then it loses control entirely over the shared key: indeed in the case of m subsequent changes the adversary would have to check more than $\binom{n}{m} > (\frac{n}{m})^m$ possibilities.

An additional advantage of this approach is that if a device A gets cloned, then a partner B of A can talk with only one version of A—evolution of keys will lead to a lack of synchronization and consequently detection of the presence of clones.

One can modify the scheme so that recovering past versions of the key becomes impossible: if F is, say, a hash function, then learning the current version of the key (e.g., by breaking into the device) does not reveal even the previous key. Despite the limited randomness in the process, key evolution has comparable properties to key unpredictability [331].

In a similar way, one can let the identifiers evolve.

1.3.5 Transmission with Errors

Transmitting the same encrypted identifiers during Phase 2 of the algorithm from Sect. 1.3.2 allows the adversary to set it as a temporary identifier for that user. To do so, the ciphertext has to be randomized. The obvious way to create non-repeatable messages is to include a nonce, namely instead of sending $c = \mathrm{Enc}_k(A)$, A selects a nonce η and transmits $\hat{c} = \mathrm{Enc}_k(A \oplus \eta)$ together with η. However, one can arrange this in a more clever way. The modified procedure is as follows:

- A chooses at random a string η with a (relatively) small Hamming weight,
- A computes $\hat{c} := \mathrm{Enc}_k(A \oplus \eta)$ and sends it to a partner (say B), with which it expects to share the key k.
- B computes $\hat{A} := \mathrm{Dec}_k(\hat{c})$ and looks for identifiers D of its partners such that the Hamming weight of $\hat{A} \oplus D$ is small. The identifier of A is among these candidates, and if the parameters are chosen properly there is only a small chance of having any other candidate on the list. Such false candidates may be eliminated easily using bilateral keys.

From the point of view of the adversary, holding a lot of keys the situation is more complicated. It may hold many false candidates for the key k used in this phase. As $\mathrm{Dec}_{k'}(\mathrm{Enc}_k(A \oplus \eta))$ provides unpredictable results, it may happen that it is close enough to some valid identifier U. In this way the adversary may get a lot of data confusing its tracing attack.

1.4 Conclusion and Future Directions

In this chapter, we focused on pointing out some of the challenges in developing large scale networks of severely constrained devices. Due to computational constraints the traditional approach to multiparty security and privacy has to give way to methods based on symmetric cryptography and information obfuscation. While considering threats caused by malicious protocol implementation, we presented several schemes utilizing an additional device, a watchdog, that may be provided by a third party, and hinders any covert channel aimed at secret leakage.

Future directions include inspecting the possibilities of leakage prevention in the case of collusion between a reader and a device. The additional device described in Sect. 1.2 prevents understanding of the messages exchanged between the verified device and the reader, however as the watchdog is not integrated into the device there is no guarantee that the device and the reader are not creating a covert channel. A natural approach would be to enable the possibility of signal jamming by the watchdog, however this solution is ineffective due to power requirements. Moreover, since each user may be in possession of multiple devices, the case of a single watchdog responsible for all a user's devices and batch authorization might be considered.

In the second part of the chapter, we pointed out the problem of violating privacy requirements, especially via user tracking, in ubiquitous systems. One of the major challenges is establishing a shared key, as 'generating' a new one with methods derived from the Diffie-Hellman protocol is not feasible and using a constant pool of predistributed keys allows a tracking adversary to identify devices during the key discovery phase. We described some methods based on a key evolution approach and on the obfuscation of information. The latter is obtained by utilizing hash functions or introducing a small error into the transmitted message. It should be noted that these solutions are not fully satisfactory, at least if we are confined to symmetric methods. If asymmetric cryptography can be used, then the situation is quite different (see Chap. 5).

Acknowledgments Authors "Mirosław Kutyłowski and Piotr Syga" supported by Polish National Science Centre, project OPUS no 2014/15/B/ST6/02837.