# Key establishment in constrained devices

Jan-Erik Ekberg

13th October 2006

## Abstract

For some classes of embedded devices or sensors the ubiquotous, well researched methods and frameworks for key establishment cannot be deployed because of device constraints. This paper examines and motivates these limitations, which also forms the design criteria for key establishment protocols related to this class of devices. A collection of existing methods are presented, and the paper is concluded with some new key establishment ideas, especially targeted for constrained, mobile devices, rather that only sensor networks.

## 1 Introduction

Key establishment between devices that communicate over an unreliable channel while still being constrained in terms of energy, cost, size or UI limitations is often problematic. This is especially true if the solution is to be tied to a radio technology rather than to a specific device (type). Typically the absolute constraints would include:

1. Few assumption can be made about the user interface. Sometimes there are no users at all (like in sensor networks), and even if there is a user, his interface can be considered restricted to a few buttons and a couple of LED:s / beeps (consider wristwatches, mice, washing machines, MP3-players).

2. The computing speed of the device is often minuscule compared to e.g. personal computers — if there is encryption on the communication channel the (symmetric) cipher suite is often a hardware solution, and its management is done with a simple, slow controller. Memory sizes of the controller can be in the single-digit kB range for code, and possibly less that 1kB of data memory.

3. The (small) batteries of the device should last for months without recharging. Power consumption issues are in some cases the foremost design constraint.

4. There is no global network support nor global connectivity. Thus, traditional internet-style key exchange and key distribution protocols based on e.g. trusted third parties (TTP:s) are not implementationally feasible.

5. Configuration during mass-production is expensive and time-consuming. Ideally, devices should leave the manufacturing line identically configured, and

any configuration should be done by the user.

6. The manufacturing cost of mass-market products is directly reflected in the end price, and is thus instrumental in the success of the design. In these segments, the benefits of the security (key establishment) must remain in proportion to any increase in cost.

In view of the constraints above, this paper considers key establishment from the viewpoint of what can be done with the tools available in the embedded devices. For clarification, asymmetric cryptography (and thus Diffie-Hellman) is discarded without further consideration, e.g. due to the cost involved of including those algorithm for the single purpose of key establishment. User involvement through full-scale keyboards and displays is also ignored, although interesting concepts can be constructed in designs where, say one device has a good display, and another a keyboard. Sensor designs form the basis of the existing work, but the intent of the paper is to move the focus somewhat more towards consumer devices.

## 2 Cost

Cost as a general concept is a main design criteria for any product, as the consumer of a product or service in the end always weighs benefit against cost, and in the face of competition, the product with the best benefit/cost ratio usually is at an advantage. Security protocols in a communication stack will induce cost for the design in several ways. If the design is to be implemented in hardware, there is a design or purchase, as well as a real estate cost (on the chip) related to the hardware block(s) that implements the security mechanism. If some security mechanisms are to be used only rarely (like in key establishment) it may make sense to apply them to the design in software, despite the much higher energy cost — on the other hand if specialized algorithms that are used only once or twice can be eliminated altogether by making use of mechanisms already deployed for some more frequent activity even larger savings can be achieved. The monetary cost will play a role during development (where the usefulness of security is weighed against the extra development cost) as well as during deployment, where the end customer will have to pay for whatever additional cost security brings.

Energy consumption is a cost issue that is especially prominent in battery-operated devices, and indirectly very noticable to the end user. For security protocols, this cost category can be split into

1. *Computational cost*: Any computation will consume energy, and the resource requirements of security algorithms are typically significant compared to other logic needed for e.g. radio transmission. Dedicated chips / hardware blocks are more energy-efficient than running the same algorithm in a general purpose controller or processor. Also, in practice the complexity of even simple security algorithms cannot perform adequately in embedded controllers.

2. *Memory cost*: Partly a subcategory of computation, the energy efficiency of memory (especially if it needs to be updated frequently) is independently of

technology fairly low if compared to HW implementing simple logic flows. So a "memory-efficient" algorithm consumes less energy than a comparably complex algorithm that needs large intermediary storage buffers. This is well presented in [10].

3. *Communication cost*: The price of a transmitted bit is a dominant factor when it comes to energy consumption — even for transmission distances in the sensor range ($\leq$ 10 m). Thus, every saved bit in communication brings down the total energy cost.

To validate the previous statements, the lower bounds of the energy cost per bit is stated by [7] to be $18\mu J/bit$, and by [11] to be $50\mu J/bit$ for their respective sensor/PAN-type radios. As transmission speed, power control, modulation and channel overhead like retransmissions and synchronization will affect the bits/J ratio for a radio to a high degree, these values can at best be considered to be real-world examples roughly indicating the range of transmission power consumption. In the comparisons below, the $18\mu J/bit$ measure is used. Regarding the difference between sending and receiving information [4], [13] or [11] all indicate that transmission is only twice as costly as receiving, or even less. At least for WLAN [4], also being in idle mode (listening for incoming packets) can be about as costly as actually sending or receiving, and although it is difficult to see the impact of security measures on this last issue, mimizing idle time from a communication viewpoint is considered e.g. in [11].

Regarding memory energy consumption, which in later measurements in many cases is included, the following deductions can be made. Both [10] and [5] cite the relative energy consumption of the flip-flop memory included in their respective cryptographic chips running at 100kHz and 3.3V / 1.5V, and the result gives a memory power consumption of 22.8nW/bit (for [10]) and 9nW/bit (for [5]) respectively. Additionally, Microprocessor cache energy consumption has been measured on an experimental basis in [6] — this can be considered an appropriate example for memory with high content volatility. Many memory technologies are reviewed, and one example (with about average powr consumption), $Dual-V_t$ memory, measures at gigabit clock speeds and 0.75V a cache energy consumption of 19.5 $nW/bit$, which is comparable to the flip-flop measures above. Thus, for a 10kB memory the last measurement amounts to 1.6 $mW = mJ/s$, i.e. energy-wise equivalent to transmitting 90 bits/s in the scenario outlined here.

The power consumption of a few common cryptographic algorithms optimized for energy are shown in table 1. The SW measurements [12], are measured on a Compaq iPaq H3670, with an Intel SA-1110 StrongARM clocked at 206MHz, i.e. on a PDA device.

| Algorithm | Energy/op (HW) | Energy/op (SW) |
|---|---|---|
| AES(128b) | 0.045 $\mu J$ [5] | 17.9 $\mu J$ |
| RSA(1024b) | 2.41/0.37 $mJ$ [8] | 546/16 $mJ$ |
| ECC(163b) | 0.66/1.1 $mJ$ [2] | 134/196 $mJ$ |

Table 1: Power consumption for some crypto algorithms)

From the table we can also make note of that

the energy consumption of all the algorithms increase by 200-400 times in their software instantiations. So, clearly, for bulk encryption / integrity checks software algorithms for security should never be used — or they easily become the dominant power sink in mobile devices. The collection table 1 compares all mentioned aspects of energy consumption:

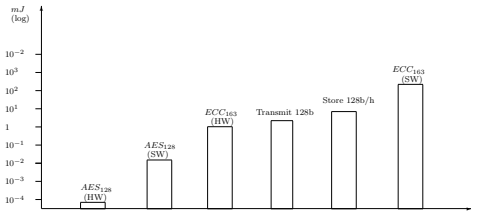


Figure 1: Energy consumption compared (logarithmic scale)

# 3 Key establishment between sensors

Key *pre-distribution* is the foremost means for key establishment in sensor networks - networks that are deployed by "dropping the sensors into enemy territory". The networks that are discussed here, although not explicitely stated in the references, assume a fixed topology, i.e. no mobility after network deployment is considered. The trivial solutions — a commonly shared key, or pairwise shared keys — are inadequate solutions because:

1. A commonly shared key, or a *mission key* for large networks suffers from the vulnerablity that a capture / break-in of any sensor might compromise the keying of the whole network, and
2. Pairwise shared keys requires storage of $n$ keys, where $n$ is the number of nodes in the network. It is a big overhead to pay in terms of configuration and memory consumption, since at deployment, only the keys for the neighbors would be used (4-5 keys out of a key set of 1000:s). The pairwise key solution also provides no extensibility of the network exceeding the original intended maximum size (spanned by the pairwise keys).

The above does not exclude the fact that these mechanisms might well be usable in e.g. small networks or in home environments where the user configures the same key into all devices. However, the further discussion is based on the notion that for some reason like network size, security requirement, or device configurability the "basic" solutions cannot be used.

Also, it is not a given in a sensor network that the security model is flat, i.e. that every device can talk securely to every other device. In fact in many cases the structure of the network itself will contain special nodes — gateways to other networks for example — that also could take a special role in a security context, as security aggregators, trusted third parties or as key servers. These kinds of constructs are however left outside of the scope of this paper.

## 3.1 Probabilistic key exchange

One of the first attempts to remedy the abovementioned problems of pre-distributed keys was achieved by probabilistic key sharing [3]. A large enough key-space, say 100000 keys is allocated, and each device is assigned

a number of keys — if say, the assigned number is 250, there is a $p$=50% probability that two randomly selected devices share a key. This $p$ can be calculated by basic probability theory, but the authors additionally refer to random graph theory to deduce whether a path exists between two nodes in the whole deployed network (a graph), in the case no shared key exists between two (neigboring) devices. It can be shown that for large graphs a threshold function exists, where this second probability moves from "nonexistent" to "certainly true" at some level of connectedness (pairwise devices that do have common keys). Thus, even if two devices do not share a key, they can find common devices — a path — over which every hop has a shared key in place, and the paper also proves that this path is rarely longer than a few hops.

The probabilistic key - sharing has the advantage, that given a large key-space additional sensors can be deployed at a later time (the network can be extended). Also, in case when a sensor is compromised, only a small set of keys will be revealed to the attacker.

The main disadvantage of the scheme is the uncertainty of the outcome, and some unclarity of how the common nodes through which pairwise security can be establised are found. The referred paper provide simulations that show path lengths of 2-16 hops for constructing the key, so protocols based on probabilistic key sharing must be fairly delay tolerant, and a quite complex network "key discovery" mechanism must be set up to accomodate for key establishment.

## 3.2 PIKE

The probabilistic exchange used the notion of local reachability for its graph model. If global addressability is possible over the deployed network, a more efficient predistributed establishment protocol exists, named PIKE (Peer Intermediaries for Key Establishment) [7]. In this work, the key space is made to fill a perfect square, every node representing one position in the $n \times n$ matrix. Prior to deployment, the node is assigned pairwise keys with all other nodes that are either on the same row or the same line as the node in question (thus $2*(n-1)$ keys will be available in the device). After deployment any two devices can construct a shared key through any node that share a row with one of the nodes and a column with the other (two such nodes exists, and are both supported for redundancy). Through the common node (with the smallest distance), using the pairwise keys that exists between that node and the nodes that wish to communicate respectively, a key is sent and the result is acknowledged.

The PIKE protocol can be extended into further dimensions (from square to cube etc,), saving storage, but the hops needed for key agreement will increase by one every time the dimensions is raised. Not all devices from the PIKE square need to be deployed at one time, making for extensibility, but the deployment must proceed in an organized fashion, so that the deployed devices constitute something close to a square at all times, and the deployed set should have no holes with respect to that order. The PIKE protocol provides perfect "reachability" in the absense of device loss.

One problem with the PIKE idea is that the common keys are scarce, and even with modest sensor loss, other, alternative paths, are needed to set up common keys. However, as a difference to probabilistic key sharing, the single PIKE node can deduce and try such paths, as a global addressing scheme is a given in the PIKE scheme.

### 3.3 Neural networks

Neural network constructs can also be used in for key establishment. A system based on tree parity machines has been proposed for this [9], and a hardware design has also been published [14]. Here, a neural network structured as a tree, has a decision function that is the sign parity of the hidden node outputs. Each device starts out with a randomly chosen weight setup between the nodes in feed-forward tree. The learning phase consists of feeding the respective devices (the neural network inputs) with the same (public) input stream, e.g. taken from a pseudo-random generator with a shared seed, to save communication. The one-bit output (1/-1) of the networks are compared (sent between the devices), and in case they differ, the individual weights of those neural network nodes that have the same parity as the end result in the device (i.e. the "wrong" value) are adjusted towards the opposite direction). The end result of this process is that the weights of the networks move towards each other, and eventually the outputs of the networks on equivalent inputs are the same. The security motivations is outlined in [9], and is the based on the idea that as the initial weights are never revealed, an eavesdropper cannot relate to the learning process. Some attacks have been proposed (listed in [14]) but seemingly suitable network parametrization will provide security against the available attacks. The use of the synchronized neural networks is as key material for another algorithm or directly as a key stream as a stream cipher.

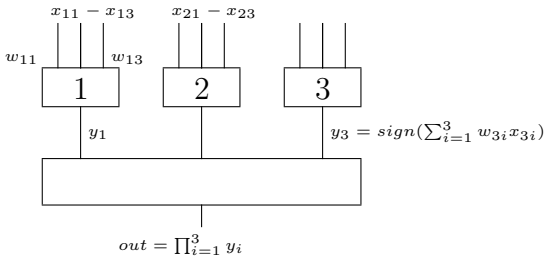


Figure 2: Tree Parity construct

It is unclear what the energy consumption or security level of the proposed neural network construct really is – at least the memory requirements might be fairly high. For personal-grade communication needs, especially in the absence of better solutions, this system provides an alternative basis for key establishment that is certainly better than no protection at all.

### 3.4 Key infection

Another take on sensor key establishment is taken by a process labelled *key infection* [1]. Here the attacker model is revised to be

1. The attacker *does not have* physical access to the deployment site during the deployment phase,

2. The attacker is able to monitor only a small proportion of the network communication during the deployment phase, and that

3. The attacker is unable to launch active attacks during the deployment phase (such as jamming)

The reasoning behind the model is that many use cases exist where networks are deployed / initiated in geographical areas where attackers are extremely unlikely to be present in large numbers until after deployment. The method of key infection roughly mean that a node broadcasts its name and key in the clear for its nearest neighborhood, and other nodes that hear this call will respond with a session key and their own identity encrypted with the broadcast key. An improved version of the system is called *whisper keying* where the broadcasting devices starts out with less transmission power, and only gradually increases the power of its brodcasts. Neighboring devices that hear the call respond with equally low transmission power. With whisper, the danger of an eavesdropper overhearing the key establishment messages is minimized.

To improve the end result and weed out attacking entities, a system named *secrecy amplification* is deployed. If two devices $W_1$ and $W_2$ have made a common shared key, but also both have established a secret with a device $W_3$, the shared key between $W_1$and $W_2$ can be amplified in the following manner: $W_1$ sends a random value to $W_2$ through $W_3$ (encrypted by the respective shared keys, $W_3$ re-encrypts), and the original shared key between $W_1$ and $W_2$ is diversified based on the random value. The end result is that if the original shared key between $W_1$ and $W_2$ was secure, the new one will also be, but even in the case where the old $W_1 \rightarrow W_2$ - key was compromised, the new one will not be if the keys $W_1 \rightarrow W_3$ and $W_3 \rightarrow W_2$ were not. Simulations indicate that the amplification improves the keying reliability by several tens of percentage points in the face of a attacker.

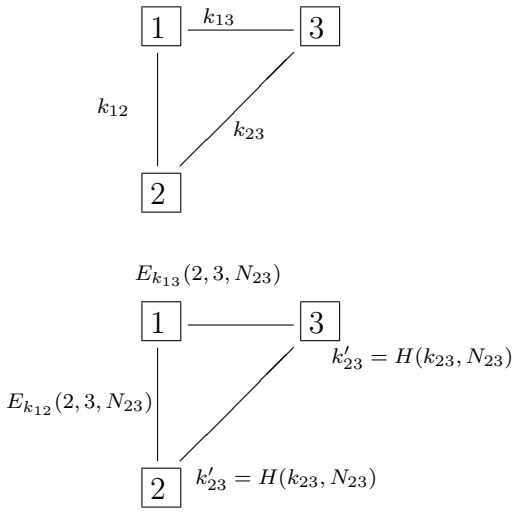


Figure 3: Security amplification

Another improvement strategy is to construct end-to-end keying. This wastes node memory, but enables multi-hop secrecy amplificaton, providing an overlay amplification strategy.

# 4 Time-domain amplification

Although the same constraints as in sensors apply, mobile devices can be seen to have one advantage over sensors - the network topology is volatile. With the same attack model used in key injection, it can be claimed that the possibility of an attacker being present at the time of key establishment is minuscule for commercial devices, possibly with the exception of the point of purchase, where

many pairings could be expected to happen. But in the spirit of secrecy amplification in the topology domain, the amplification in the mobile scenario can be executed in the time domain. By asking the customer to keep the devices with him (e.g. in a pocket) for an hour up to a day, it is unlikely that an eavesdropper would be able to "follow the pocket" reliably if the user moves around. Trivially, secrecy amplification can be acheived by the following protocol

```
D1                          D2

           RAND_A
------------------------->
           RAND_B
<-------------------------

    k_1 = E(RAND_A, RAND_B)

          ... time ...

    E(k_1, RAND_A2),E(k_1, D1, D2, RAND_A2)
------------------------->
    E(k_1, RAND_B2),E(k_1, D2, D2, RAND_B2)
<-------------------------

k_2 = E(k_1, RAND_A2 XOR RAND_B2)

          ... time ...
```

where $E$ denotes an encryption function or a keyed cryptographic hash — in fact the one-wayness of a hash may eliminate some possible attacks. The user can be alerted if the pairing fails at some stage, as well as stopped (and even continued) at any suitable time. This protocol has the partial advantage that the devices may be idle and saving energy during the time intervals, but the number of rounds will tax energy consumption (however, compared to a software implementation of e.g. ellipitic curve Diffie-Hellman (ECC/DH) hundreds of messages can be exchanged to be equivalent in energy consumption). The protocol is not suitable for adhoc (quick set-up) connections, and neither for situations where one of devices is fixed (like a washing-machine).

# 5 Fixed-device amplification

The original amplification strategy in [1] can be made to work in the mobile domain by adding a cache to the amplification messages. Consider devices $D_1$, $D_2$ and $D_3$ where $D_1$ is a fixed device, and $D_2$, $D_3$ mobile. These devices all have paired previously, so keys $k_{12}, k_{13}$ and $k_{23}$ exist. Still the recent key $k_{12}$ is waiting to be amplified. Now, when $D_2$ meets $D_3$, it will give a list of peers that it wants amplification for, amongst those $D_1$. As $D_3$ has a key for $D_1$, it will accept a packet $E_{k_{23}}(D1, D2, N_1)$ and return the reply to $D_2$ in the form $E_{k_{13}}(D1, D2, N_1), D3$. Now $D2$ can carry this cached amplification back to $D1$, and if $k_{13}$ has not changed in between (possibly the key between $D1$ and $D3$ is also being amplified), the amplification can be carried out, and a new key (e.g. $k'_{12} = E_{k_{12}}(N_1)$) can be constructed and validated between devices $D_1$ and $D_2$. The end result of the amplification is equivalent to the original case - if $k_{23}$ was safe, then $k_{13}$ will now also be, otherwise no change occured in the secrecy.

The strategy of cached amplifications need not be restricted to fixed devices, it can also be used for any device in the network to improve key confidentiality.

# 6 Radio environment info amplification

A commonality that is present in peer-to-peer mobile networks is that devices will make the presence known to peers for the sake of enabling incoming connections. This strategy is often named *advertising*. Considering the amplification algorithm in the time domain, the initial assumption being only that the attacker cannot follow the mobile devices long enough to keep track of the pairing process. Additionally the devices that are establishing the key can make use of the changing environment by making a log of device addresses that are visible during the intervals where the key establishment process is idle. Because the list of visible device might not be end up to be equivalent, the following strategy could be used (this sequence is combined with the actual key amplification):

```
     D1                        D2

           ADDR_1/LOW_BITS
     ------------------------->
                           if ADDR_1 found
                              k' = E(k, ADDR_1)
                           else k' = k

          ADDR_2/LOW_BITS,
        E(k',ADDR_2/LOW_BITS)
     <-------------------------
determine k'
if ADDR_2 found
 k'' = E(k', ADDR_2)
else k'' = k'
```

where the lengths of the address parts and checksums can be set to quite short (2 bytes each) as the protocol will catch ambiguity and protection against active attacks (like the attacker actually producing most of the visible address content) is limited. Also collisions in LOW_BITS can simply be ignored — treated as addresses that are not seen.

Equivalently, any other contextual information that is available to the pairing devices (service discovery, satellite beacons, temperature, wind, ...) can be used in the same manner as the encountered addresses.

# 7 Obfuscation in noisy environment

A basic form of security augmentation by means of obfuscation is Merkle's puzzles (see figure 4). There one device transmits a number of puzzles (say $N$) to the other party, that are solvable in some limited time, and by solving a puzzle an index, and a key are revealed to the receiver. The puzzles in the picture are solved by trying all possible keys until the $CONST$ parameter is revealed in a decryption, if so, the index ($RAND_x$) and the key ($F(RAND_x)$) likewise. From the received puzzle set, the receiver solves one puzzle, returns the index, and the corresponding key can be retrieved by the sender based on the index. An attacker has no use of the transmitted index, so he must break on average $N/2$ puzzles to find the index and the corresponding key. As the attacker is faced with a more difficult computational problem some measure of security can be attributed to this scheme. To be noticed, however, is that the increase in work is linear.

As a sideline, another aspect that is always controllable by a radio layer is the addressing. Although the device (MAC) address usually is fixed and globally unique, this is not at all necessary. The confusion with global addressing is well stated in [**?**]: The address is currently used 1) as a

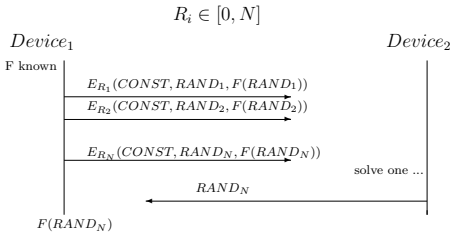


Figure 4: Merkle's Puzzles

unique identifier "in the entire" world to index sessions (like shared security contexts) and identify devices, as well as 2) a local identifier to distinguish between devices in the layer 2 network — which in e.g. PAN networks often consists of less than 10 nodes! Thus for local addressing the fixed and globally unique identifier is no requirement — although a convenient identity mechanism, there are other (and cryptographically better ways) to assert identity, e.g based on user credentials.

Now, consider an environment where devices come and go, and devices also might change their addresses at will, e.g. for privacy reasons. So there is a lot of change going on, if the device periodically scans its environment.

We also assume an input mechanism on the devices involved in the pairing. This input might constitute a single button. The direction of the key establishment is determined (one device transmits the key, the other one listens). The algorithm can be interleaved in a way where both devices send the key in alternating button presses or even, at the expense of security, in parallel (no order).

Returning to the concept of Merkle's puzzles, the whole pairing process (which in an ideal case could take hours) requires both devices to advertise random addresses changing at random intervals. Here, the choice is between a deployment strategy where the random address is changed very frequently, but within scanning resolution (so that an active scanner would with a very high probablity catch all advertised addresses), or an adaptive strategy where the random address update is adjusted to the current volatility of the environment. In the second case, the change of address also implies that the broadcasting of the new address need not follow immediately after the disappearance of the old, but rather this interval is also adjusted to environmental conditions. The basic difference between the strategies is that in the first instantiation the devices strive to produce "false" information as much as possible, while the attacker is given the means to identify when a pairing takes place, whereas the second strategy hides the pairing in an already noisy environment, with the additional benefit of saving power. The latter strategy is to its advantage in public places, whereas the first excels in secluded radio environments (like in homes).

The actual key establishment proceeds as follows: The sending device randomly generates twp random keys $key$, and $key_{trans}$ (say 128 bits each), and divide both into $N$ (say 16) pieces, each piece being one byte in our example. The $key_{trans}$ is diversified into 16 separate keys (see figure 5), and for each piece of the key $key[i]$, a pattern $pat_i$ is produced by encrypting with the diversified key: $pat_i = E_{kt_i}(key[i])$. 16 transmission packets are constructed by appending as much of $pat_i$ as possible to $key_{trans}[i]$ to form the length of an address in the wireless system.

After the initialization phase, the key agreement / transfer can start. The user is asked

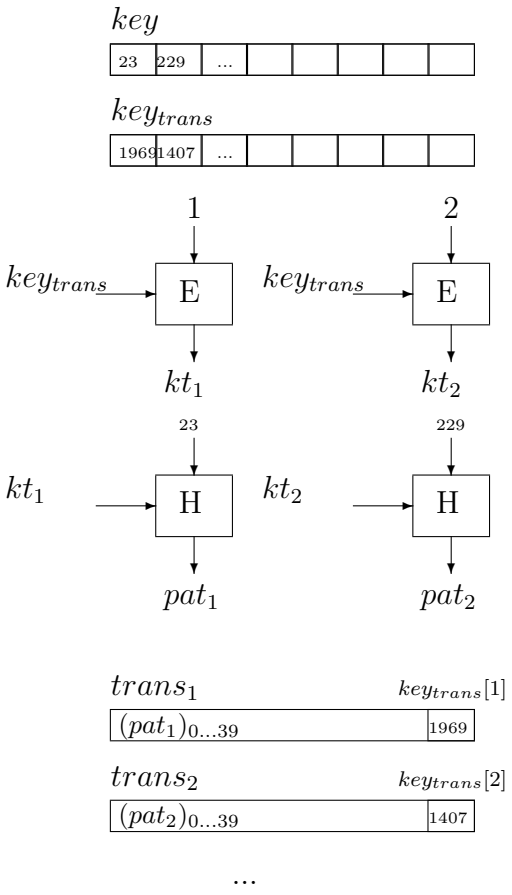


Figure 5: Noisy Algorithm

to press the buttons on both devices simultaneously. When the key is pressed the sender changes its own address to the relevant transmission packet. The receiver will take note in its scan which new (not previously seen addresses) appeared during a short interval (+/- 3 seconds?). This implies that the receiver should be continuosly scanning during the pairing process, and maintain some form of FIFO queue of recently seen addresses.

The user should press the buttons N times, preferrably with some interval, and possibly movement in the intermission. Note that these events are subliminal in the face of an observer on the radio alone - there is no clear indication when a pairing starts, and when it ends.

After the last button press, the receiving device has collected a path (possibly even alternative paths) of devices that happened to appear at the right moment(s). It is, however, preferrable, that the devices continue changing addresses for a random period also after this event. From these paths, a number of alternative keys $k$ of the counterpart can be retrieved. The keys are tested against the addresses of the path, the *patterns*, revealing the $x_i$:s if a match is found in the addresses. This is the equivalence of the puzzle in Merkle's puzzle. The concatenations of the $x_i$:s form a key.

The critical augmentation, and the reason why this puzzle is harder — difficulty becomes exponential in relation to the number of transmitted key parts in addition to the $O(N)$ increase for the total transmitted material is that a hidden function provides synchronization between the sender and the receiver, invisible to the attacker.

This system also lends itself to password initiation. E.g. a 4-digit PIN can be labeled on a headset. The PIN is transformed (by any pseudo-random generator) into a timed sequence of N events (the timing need not be more exact that in the range seconds to an hour). On entry into a phone, the same sequence can be initiated, and the process can be that an unpaired headset starts the pairing at boot-up, and the phone user is asked to synchronize to that, replacing the button-presses. However, care must be taken not to let the resolution of the PIN affect the resolution of the timing.

For a short calculation of the attack strength in the face of two secluded devices, where the devices change addresses on average every two seconds, and the user actually follows instructions and presses the buttons e.g. eight times during a 5-minute "coffee break". Both devices will transmit 150 addresses, i.e. 300 addresses will be visible on the radio. As there is no indication of which address belongs to which device the stategy is to try all combinations of 16 addresses in the face of the 300 presented ones, which equals

$C = \binom{300}{16} \approx 10^{22}$

As the key is confirmed by, on the average 128 tries with AES, the end result amounts to sigificant work for the attacker. Even for the legitimate device the workload becomes heavy if the path space grows. E.g. if assuming an energy-efficient AES chip that ciphers one block in 10 ms in the embedded devices the transfer should be repeated if too many path choices present themselves to the legitimate receiver. (leading to an easy denial-of-service attack). Also, the direction of the tranmission, if at all possible should be directed from the less poweful device (producing the key) to the more powerful device (resolving the key).

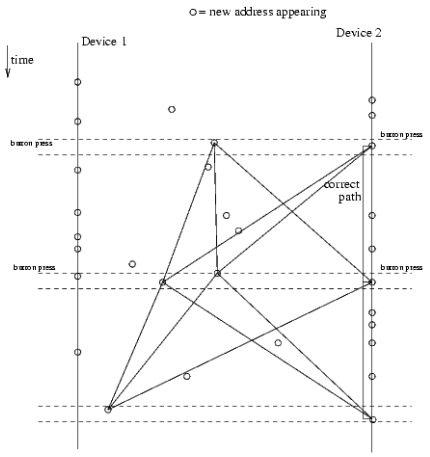


Figure 6: Path problem for the legitimate receiver

In the general case, the problem can be reduced to combinatorics over the time slots, and for fixed timeslots other solutions (even without obfuscated messages/addresses that essentially serve as synchronization) can be envisioned. The transmission key need not be transferred, but can be derived e.g. from timeslot sequence numbers.

One critical thing to note in the general case is that the workload of the attacker can be reduced to a single encryption per try, if the transmission key is straight-forwardly used for encryption and some guess can be made regarding the content at some specific instance – the attacker takes his puzzle from the use of the key encryption and not from the puzzle. The current solution provides this protection at a computational cost — a fast re-negotiation of a new key should be equivalently secure.

This algorithm is not energy-efficient, in fact in its first instantiation it is wasting energy

to provide obfuscation, and through that, security. It is, however, cost-efficient in the presence of symmetric encryption (or keyed hash) hardware, the key establishment logic is nearly trivial and easy to implement.

In this scheme, the enlightened behaviour of users is of importance. Hurrying through the scheme will make an attack significantly easier. easier. The instructions in a mobile scenario should read: "Start the pairing, and keep the devices close to each other in a pocket, handbag or equivalent. Go for a coffee or the bus to work, and now and then dig up the devices and press the buttons simultaneously. You'll have to press the buttons 16 times (a beep will sound when you press the buttons the 16th time). Take your time, the longer you keep at it, the better security you will end up with".

# 8 Acknowledgements

I thank the reviewers — N.Asokan, Dan Forsberg and Jani Suomalainen — for suggesting improvements to the solutions presented in this paper. I also thank Kaisa Nyberg for useful comments during the writing of this paper that greatly improved the presentation of this paper.

# 9 Conclusions

In parallel with more efficient versions of mainstream security (e.g. ECC chips), wireless communication, e.g. through RFID technologies, is hosted by evermore smaller and cheaper devices (advertisments, name tags, ..). Whereas communication confidentiality and integrity is relatively easy to arrange for communication between these kind of devices, in the problem area of key establishment simple solutions, tailored to specific communication domains and using available cryptographic services will remain more cost effective than mainstream pairing algorithms. This paper adds to this toolset by summarizing some of the design criteria present in the domain, and presenting a few new ideas and methods for establishing key material in truly constrained devices.

# References


[1] Ross Anderson, Chan Haowen, and Adrian Perrig. Key infection: Smart trust for smart dust, 2001.

[2] Guido Bertoni, Luca Breveglieri, and Matteo Venturi. Ecc hardware coprocessors for 8-bit systems and power consumption considerations. *itng*, 0:573–574, 2006.

[3] L. Eschenauer and V. Gligor. A key management scheme for distributed sensor networks, 2002.

[4] M. Feeney, L.M.; Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE , vol.3, no.pp.1548-1557 vol.3*, 2001.

[5] J.; Rijmen V. Feldhofer, M.; Wolkerstorfer. Aes implementation on a grain of sand. In *Information Security, IEE Proceedings , vol.152, no.1pp. 13- 20, Oct.*, 2005.

[6] V Agarwal S Keckler D Burger H Hanson, S Hrishikesh. Static energy reduction techniques for microprocessor caches. *IEEE Transactions on VLSI Systems*, 11(3), June 2003.

[7] A Haowen Chan; Perrig. Pike: Peer intermediaries for key establishment in sensor networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE , vol.1, no.pp. 524- 535 vol. 1, 13-17 March*, 2005.

[8] A.P. Charndrakasan J Goodman. An energy-efficient reconfigurable public-key cryptography processor. *IEEE Journal of Solid-state Circuits*, 36(11), Nov 2001.

[9] I. Kanter, W. Kinzel, and E. Kanter. Secure exchange of information by synchronization of neural networks. *Europhysics Letters*, 57:141, 2002.

[10] C Rechberger M Feldhofer. A case against currently used hash functions in rfid protocols. Workshop on RFID Security 2006 - RFIDSec06, July 13-14, Graz, Austria.

[11] N.H. Miller, M.J.; Vaidya. Minimizing energy consumption in sensor networks using a wakeup radio. In *Wireless Communications and Networking Conference, 2004*, 2004.

[12] Nachiketh R. Potlapally, Srivaths Ravi, Anand Raghunathan, and Niraj K. Jha. Analyzing the energy consumption of security protocols. In *ISLPED '03: Proceedings of the 2003 international symposium on Low power electronics and design*, pages 30–35, New York, NY, USA, 2003. ACM Press.

[13] Tijs van Dam and Koen Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 171–180, New York, NY, USA, 2003. ACM Press.

[14] Markus Volkmer and Sebastian Wallner. Tree parity machine rekeying architectures for embedded security.