Secure Group Communication in Ad-Hoc Networks using Tree Parity Machines

Björn Saballus[1], Markus Volkmer, and Sebastian Wallner[2]

[1] System Architecture Group, University of Karlsruhe, D-76128 Karlsruhe, Germany
saballus@ibds.uka.de***
[2] Institute of Computer Technology, Hamburg University of Technology, D-21073 Hamburg, Germany
{markus.volkmer,wallner}@tuhh.de

Abstract. A fundamental building block of secure group communication is the establishment of a common group key. This can be divided into key agreement and key distribution. Common group key agreement protocols are based on the Diffie-Hellman (DH) key exchange and extend it to groups. Group key distribution protocols are centralized approaches which make use of one or more special key servers. In contrast to these approaches, we present a protocol which makes use of the Tree Parity Machine key exchange between multiple parties. It does not need a centralized server and therefore is especially suitable for ad-hoc networks of any kind.

1 Introduction

Most group communication takes place over vulnerable open networks like Wireless LAN. To secure the communication, the sent data needs to be encrypted. This can be done by asymmetric or symmetric cryptography. Asymmetric cryptography has some disadvantages for group communication. It requires each group member to hold the public keys of every other member in the group. If a new participant wants to join, all current group members need to get the corresponding public key. Additionally they have to send their own public keys to the new participant. Even more costly, whenever one member of a group of size n wants to send data to the rest of the group it has to encrypt the data $n-1$ times and send $n-1$ differently encrypted data sets. Therefore broadcast/multicast is impossible as the described sending pattern is equivalent to a unicast transport.

To be able to use broadcast messages to decrease the number of send messages and needed bandwidth for the key exchange, this leads to the use of symmetric cryptography. Here, all members share the same secret group key. This group key needs to be exchanged and managed in some way, which is addressed as *group key management* in the literature. A contemporary overview is given in [1]. To provide the key material to all participants or members in a group is the main

*** Work done while being with Hamburg University of Technology.

problem of *secure group communication*. It is addressed as *key management*, which in turn is divided into *key agreement* and *key distribution*.

Especially the vision of Ubiquitous and Pervasive Computing ([2,3]) multiplies the demand for secure group communication, as (on principle) 'everything will be connected' and mobile nodes as well as frequently changing network topologies impose demanding requirements on any security solution. Key exchange and entity authentication are of major importance with regard to security [4]. The often present lack of infrastructure penalizes approaches needing a central authority securely providing public keys.

In order to avoid the necessity of a public key infrastructure in asymmetric cryptography approaches, the symmetric (secret-key) DH key exchange protocol [5] represents the basis for many secure group communication protocols. Elliptic Curve Diffie-Hellmann (ECDH) is applied in [6] to establish a secure end-to-end connection between a sensor and a base station, for example. Other approaches for DH-key agreement in peer-to-peer wireless networks requiring an initial physical contact or at least a geographic proximity are inspired by the work of Anderson et al. [7]. In [8], direct user interaction in terms of some manual or visual confirmation for authentication is proposed.

We describe the use of Tree Parity Machines [9] to establish a common group key. We present a protocol which belongs to the key agreement approaches. It does not need a centralized server or any special group members which perform special tasks. In contrast to most other protocols in the literature, it is not based on the extension of the DH key exchange protocol [5] to groups. The protocol is especially applicable for ad-hoc networks of any kind in which no server is present and members can join and leave the group communication at any time.

Note that within this work, it is assumed that all participants are already authenticated to join the group and take part in the key exchange. A straightforward entity authentication inherent in the Tree Parity Machine algorithm is described in [10]. It is therefore not required to implement an additional (maybe complex) mechanism. Also, senders and messages are trusted. Furthermore, we do not consider routing in this work and no routing algorithm is implemented. Of course, several routing protocols for ad-hoc networks have been proposed. Some of these protocols are for example the *Scalable Source Routing (SSR) [11]*, *Ad-hoc On-Demand Distance Vector Routing (AODV)*[12] or *Dynamic Source Routing (DSR)*[13]. As other key distribution protocols, our protocol will have to deal with the routing issue as well.

2 Tree Parity Machines and Group Key Exchange

The fast synchronization of interacting identically structured Tree Parity Machines (TPMs) is proposed by Kinzel and Kanter [9] as a method for symmetric key exchange. It does not involve large numbers and principles from number theory and can be used for a low-cost hardware realization [14,15]. The exchange protocol is realized by an interactive adaptation process between the two interacting parties A and B. The TPM (see Figure 1a) consists of K independent

summation units ($1 \leq k \leq K$) with non-overlapping inputs in a tree structure and a single parity unit at the output.

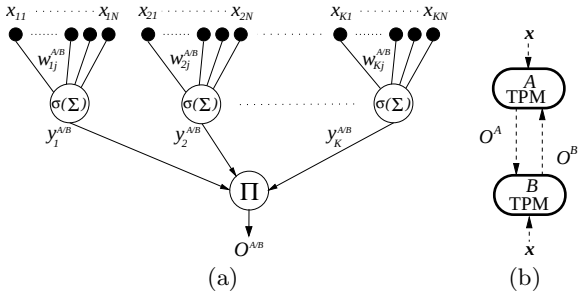


Fig. 1. (a) A Tree Parity Machine. A single output is calculated from the parity of the outputs of the summation units. (b) Outputs on commonly given inputs are exchanged between parties A and B for adaptation of their preliminary key (their internal state $w_{kj}^{A/B}$).

Each summation unit receives different N inputs ($1 \leq j \leq N$), leading to an input field of size $K \cdot N$. The vector-components are random variables with zero mean and unit variance. The common random inputs x_{kj} can also be kept secret between the parties, yielding entity authentication, as will be further explained in Section 2.1.

The output $O^{A/B}(t) \in \{-1, 1\}$ (A/B denotes equivalent operations for A and B), given bounded coefficients (weights) $w_{kj}^{A/B}(t) \in [-L, L] \subseteq \mathbb{Z}$ (from input unit j to summation unit k) and common random inputs $x_{kj}(t) \in \{-1, 1\}$, is calculated by a parity function of the signs of summations ($\sigma(\cdot)$ denotes the sign-function):

$$\begin{aligned} O^{A/B}(t) &= \prod_{k=1}^{K} y_k^{A/B}(t) \\ &= \prod_{k=1}^{K} \sigma\left(\sum_{j=1}^{N} w_{kj}^{A/B}(t)\; x_{kj}(t) \right) . \end{aligned} \tag{1}$$

2.1 Synchronization and Key Exchange

The so-called *bit package* variant (cf. [9]) reduces transmissions of outputs by an order of magnitude and also increases the security. It is thus advantageous for practical communication channels with a certain protocol overhead.

Parties A and B start with an individual randomly generated secret initial vector $w_{kj}^{A/B}(t_0)$. These initially uncorrelated random variables become correlated (identical) over time through the influence of the common inputs and the

interactive adaptation as follows. After a set of $b > 1$ presented inputs, where b denotes the size of the bit package, the corresponding b TPM outputs (bits) $O^{A/B}(t)$ are exchanged over the public channel in one package (see Figure 1b). The b sequences of signs of the summation units $y_k^{A/B}(t) \in \{-1, 1\}$ are stored for the subsequent adaptation process.

A Hebbian learning rule adapts the coefficients (the preliminary key), using the b outputs and b sequences of signs. They are changed only on equal output bits $O^A(t) = O^B(t)$ at both parties. Furthermore, only coefficients of those summation units are changed, that agree with this output, i.e. $O^{A/B}(t) = y_k^{A/B}(t)$:

$$w_{kj}^{A/B}(t) := w_{kj}^{A/B}(t-1) + O^{A/B}(t)\; x_{kj}(t)\;. \tag{2}$$

Coefficients are bound to remain in the maximum range $[-L, L] \subseteq \mathbb{Z}$ by reflection onto the boundary values. This is necessary for the synchronization and also implies that the sum in Eq. 2 cannot be inverted.

Iterating the above procedure in an interactive protocol, each component of the preliminary key performs a random walk with reflecting boundaries. The resulting key space is of size $(2L+1)^{KN}$. Two corresponding components in $w_{kj}^A(t)$ and $w_{kj}^B(t)$ receive the same random component of the common input vector $x_{kj}(t)$. After each bounding operation, the distance between the two components is successively reduced to zero. Thus the non-linear mapping from the inputs to the output bit (Eq. 1) is also changed in each step depending on the interaction of the two parties and their secret random initial coefficients. Parties with identical inputs always converge to a dynamic common trajectory. Parties with differing inputs x_{kj} always diverge.

When both parties adapted to produce each others outputs, they remain synchronous without further communication (see Eq. 2) and continue to produce the same outputs on every commonly given input. Common coefficients are now present in both TPMs in each of the following iterations. This preliminary key can be used to derive a common time-dependent final key or can be used directly.

The fact that, once synchronized, the TPMs produce equal outputs and all their internal states are identical can be used to produce subsequent keys by each TPM learning with its own output. This so-called *trajectory mode* allows to generate new common keys without further communication.

Furthermore, synchrony is achieved only for *common* inputs. Thus, keeping the common inputs secret between A and B can be used to have an (entity) authenticated key exchange. The outputs are public. Yet, an attacker that does not produce the same inputs modifies his coefficients differently when applying Eq. 2 with a different input. He so performs an adaptation different from two parties that know the common secret. He cannot synchronize and thus not exchange a common key. This principle is further described in [10]. There are $2^{KN} - 1$ possible inputs in each iteration, yielding as many possible initializations for a pseudo random number generator as a common secret for authentication.

As a test for synchrony cannot practically be defined by checking whether weights in both nets have become identical, one tests on successive equal outputs

in a sufficiently large number of iterations t_{min}, such that equal outputs by chance are excluded:

$$\forall t \in [t', \cdots, t' + t_{min}] : \; O^A(t) = O^B(t) \; . \tag{3}$$

Synchronization time is finite for discrete weights and peaked around 400 output exchanges for the parameters and the learning rule given in [9].

2.2 Group Key Exchange

Multiple parties or groups can exchange a common key based on TPM interaction and the synchronization property as proposed in [16]. Once two parties p_1 and p_2 have synchronized and thus exchanged a common key, they have identical internal states w^{p_1/p_2} and can be considered a single (identical) TPM $p_{1,2}$. This property can be exploited to derive a group key exchange, by grouping identical TPMs and adapting them simultaneously.

The exchange of a common key between $G > 2$ parties can be achieved by two basic strategies: parallel interaction processes and sequential interaction processes. Without loss of generality an appropriate numbering (and renumbering) of parties can be performed.

Parallel Interaction Processes Using parallel interaction processes, an even number of parties G is initially divided into k groups of interacting pairs $(p_i, p_j)_k$ with $k = 1, \cdots, G/2$, $i, j = 1, \cdots, G$ and $i \neq j$, performing a pairwise (independent) key exchange in parallel as explained before. After each group has a common key, pairs of synchronous groups now interact again (in a divide-and-conquer strategy) to exchange a common key (see Figure 2a).

This is done until two remaining groups synchronize the final common key in a final interaction process:

$$\begin{aligned} & (p_1, p_2)_1, (p_3, p_4)_2, \cdots, (p_{G-1}, p_G)_{G/2} \\ \rightsquigarrow \; & (p_{1,2}, p_{3,4})_1, (p_{5,6}, p_{7,8})_2, \cdots, (p_{(G/2)-1}, p_{G/2})_{G/4} \\ \rightsquigarrow \; & \quad \cdots \quad \rightsquigarrow p_{1,\cdots,G} \end{aligned} \tag{4}$$

If G is odd, the remaining party waits until all other $G - 1$ groups have exchanged a common key and then performs one last interaction process with the synchronized group. The complexity of this multi-party key-exchange scales logarithmic with the number of parties, i.e. $O(log\ G)$.

Note that the parallel variant requires either independent parallel or multiplexed communication channels. Also note that in a practical implementation only two TPMs in each group have to actively send and receive output bits, whereas the others in the group only receive.

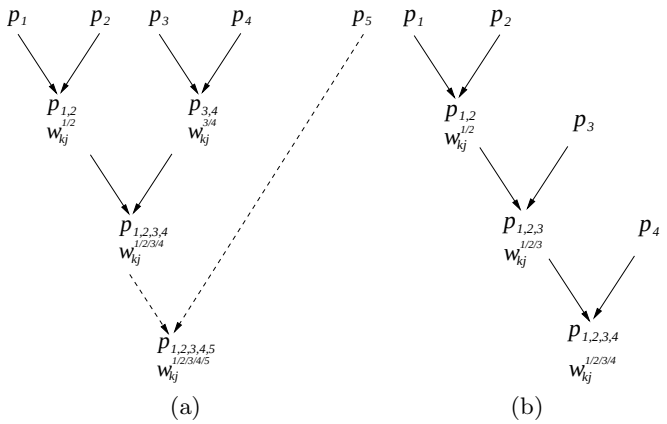


Fig. 2. Example of the two interaction principles for group key exchange. (a) Parallel Interaction Processes. If the number of parties is odd, the last party synchronizes with the group in a single (sequential) process. (b) Sequential Interaction Processes. As in the parallel variant, there is always only one group sender. All other members in an established (sub-)group only receive output bits.

Sequential Interaction Processes In a sequential interaction processes, two parties p_1 and p_2 exchange a common key as described before. Having a common key they become a group $p_{1,2}$ that now interacts with a third party p_3, and so on (Figure 2b). This way, a linear chain

$$\begin{aligned} & (\cdots((p_1, p_2), p_3), \cdots), p_G) \\ \rightsquigarrow\ & (\cdots((p_{1,2}, p_3), p_4)\cdots), p_G) \\ \rightsquigarrow\ & \cdots \rightsquigarrow p_{1,\cdots,G} \end{aligned} \tag{5}$$

of interaction processes is performed. Note that for the group only one sequence of outputs has to be communicated, as it is identical to all parties (TPMs) in the group.

Again, in practice only one TPMs in the group has to actively send and receive output bits, whereas the others in the group only receive. The complexity of this multi-party key-exchange scales linear with the number of parties, i.e. $O(G)$.

As each key exchange process (parallel or sequential) can independently be attacked, the security in the presented multi-party scenario scales inversely proportional to the number of parties.

In contrast to other group key exchange protocols, we designed a protocol which includes the group management as well. This group key exchange protocol is divided into two phases:

1. The *initialization phase* and
2. the *synchronization phase*

as described above.

The *initialization phase* is responsible for setting up the communication environment. All participants who want to join in the key exchange claim their preparedness and identify themselves to each other. Afterwards, the *synchronization phase* is setting up the group structure, performs the TPM processes and manages the output exchange and the communication.

To limit the traffic and needed bandwidth, multicasting is used to exchange the TPM output bit packages. As multicast requires the use of the UDP protocol (which is connectionless and unreliable) it is not guaranteed that the packages will be delivered to all recipients. Therefore, all previous bit packages are summarized and sent together with the current one. This history of all bit packages gives each participant the chance to catch up with the synchronization process if a previous packet was lost.

3 Tests and Results

The primary goal we focused on is a proof of concept that the group key exchange using TPMs is possible. We conducted tests in three different environments. A software simulation was used to investigate the TPM group key exchange functionality. Besides the basic functionality, the synchronization time growth was investigated.

The final software implementation was used to run tests in two different environments. One test environment was a Linux cluster connected via a LAN to allow statements for networks with up to 10 participants. The other test environment consisted of different laptops connected through a WLAN. These laptops were set up to form an ad-hoc network. We think this is enough for a proof of concept, but we are well aware that this is not sufficient for final statements about the usability. Therefore, tests in a network simulator like OMMNET or NS2 have to be made.

In the following, the parameters for the TPMs are $K = 3$ (hidden units), $L = 4$ (weight vector element range), $N = 11$ (number of inputs per hidden unit) and a bit package size of 32. The key size is defined as $K \cdot N \cdot L$, i.e. 132 bit ([14]).

3.1 Software Simulation

For a first validation of the TPM group key exchange functionality a software simulation was written. Its settings are the number of group members, the synchronization mode and the number of group key exchanges to be performed. The tests are run with a group size of two up to twelve members. For each group size 10000 group key exchanges were made.

The result parameter is the total number of exchanged bit packages ρ needed to establish a common group key. To compute this value, the maximum number of needed bit packages for each synch run are summed up.

The number of bit packages and not the synchronization time was used as a measure because the simulation was run on a single computer. The average of the different test runs, with 10000 group key exchanges each, are computed and plotted in Figure 3. The results plotted in this figure agree with the expectations: It is observed that the curve for the parallel approach (upright crosses) stays beneath a logarithmic curve and a runtime of $O(log(n))$ is approximated. The curve for the sequential approach (tilted crosses) is linear with a runtime of $O(n)$.

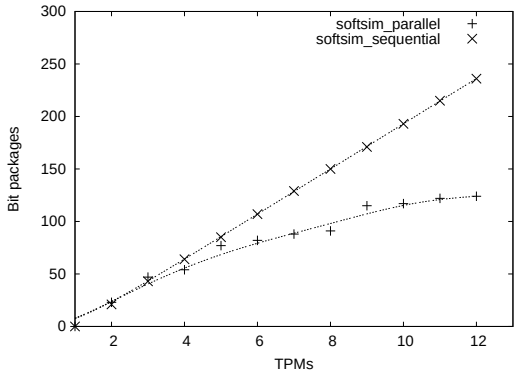


Fig. 3. Results of a software simulation for parallel (upright crosses) and sequential (tilted crosses) mode (132 bit key). The case for two parties is depicted for comparison.

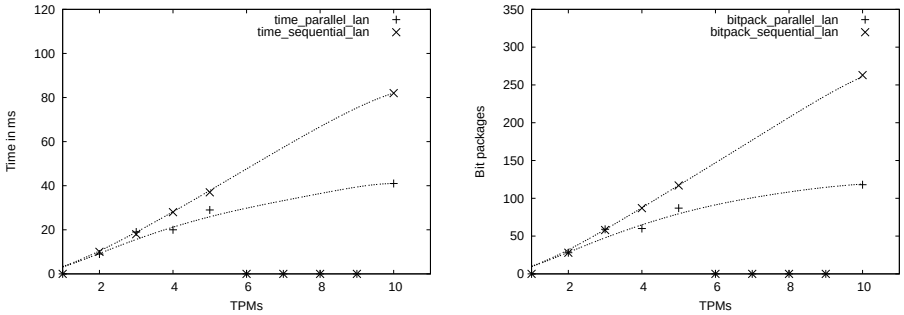
The results for the parallel mode are worth a closer investigation. For those tests with the same number of Synchronization Runs (SRs) but different numbers of participants, (e.g. the test with 3 and 4 participants each having 2 SR per key exchange) the values are slightly higher for a greater number of participants. The reason is the higher number of synchronizations per SR. As described in section 2, the number of output exchanges needed until synchronization is reached is distributed. Therefore, successful synchronizations which are performed in parallel have different numbers of output exchanges.

3.2 Network Tests

In the following, the tests made in LAN and WLAN environments are described. The developed software is a flexible cross-platform network solution written in C. It runs on Windows as well as on Linux and can be used in LANs as well as in WLANs.

LAN Test under Linux: This test is run on a Linux cluster. Each node is started on its own workstation. The connection is established using the 100 $Mbit/s$ Ethernet interface. This scenario is chosen to generate results for groups with more

than three participants and to demonstrate the scalability in a wired network environment. Each test scenario processes 500 group key exchanges in a row. Four test runs with a group size of two up to five participants each are made. No real difference can be seen in the synchronization time or number of bit packages for the given sizes. Therefore an additional test run with 10 participants was made to be able to compare the results with the software simulation and against each other.



(a) (b)

Fig. 4. Summary of the Linux wired network test. (a) Average time needed for synchronization. (b) Average number of packages needed for synchronization. Again the key size is 132 bit and the case for two parties is depicted for comparison.

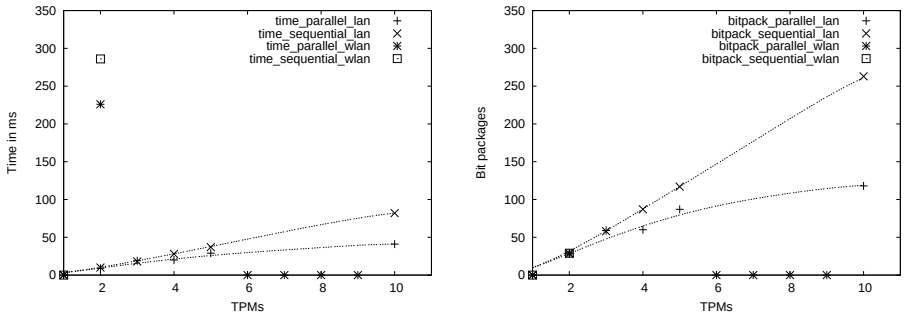
We measure the average value of bit packages and the average value of time needed for each test run. The time is measured only for the synchronization phase. The results for the run time are shown in Figure 4a and for the bit packages in Figure 4b.

Additional Test: Additional tests are made to test the *robustness* of the implementation. These tests are performed in the LAN as well as in a WLAN environment. The WLAN environment is set up using three laptops running Windows.

- *Join, leave and quit:* To test the join, leave and quit operations, the program was running on different computers (workstations or laptops) and the key exchange was started. After the first key exchange finished successfully, it was tested if new participants can join the group and if any member can leave the group without any problems or failures. This test was successful.
- *Participant breakdown:* Next, a *participant breakdown* was simulated and tested. To simulate this failure, the key exchange is started. During the initialization or synchronization phase, one or more parties were terminated. Now, the remaining parties are waiting for a packet from the missing participant. This is correctly detected after a predefined time by one or more

parties. Which and how many parties detect a missing participant depends on the mode and the member that was terminated. No problem occurred during this test, neither in parallel nor in sequential mode. It does not matter which participant was terminated. It was successfully eliminated from the list and the key exchange finished with a common group key.

In Figure 5 the average values generated during the LAN and the two party WLAN tests are compared.



(a) (b)

Fig. 5. Comparison of the Windows WLAN and Linux LAN network test. (a) Average time needed for synchronization. (b) Average number of bit packages needed for synchronization. Again the key size is 132 bit and the case for two parties is depicted for comparison.

The key exchange protocol is designed to handle member breakdowns or lost packets and precautions have been taken to deal with those failures. During the LAN tests, no participant had to wait for a packet, and therefore it can be assumed that no packet was lost. The WLAN test shows for the sequential mode, 17 events, and for the parallel mode, 37 events in which one member was waiting a specific time for a packet from the other side (see Table 1).

As the waiting time for a packet is set to one second, this explains the higher synchronization time for the WLAN tests. When this time was up, the problem was addressed and the synchronization continued successfully. As this never happened in the LAN environment, we assume the reason lies in lost packets and not in any malfunction.

To further investigate the timing, the results are cleaned from those runs, in which the waiting occurred. This means that all runs with a significant higher synchronization time than one second have not been considered. Without these events, an average synchronization time, for the sequential mode of about 180 ms and for the parallel mode of about 183 ms per key exchange, are computed.

Furthermore, the packet order is investigated. In the WLAN network, it can be observed that for sequential mode 3, and for parallel mode 0, packets arrived

2 TPMs / 500 runs	LAN		WLAN	
	seq.	par.	seq.	par.
Average number of bit packages	28	28	29	29
Average time in ms	10	9	286	226
Gross bandwidth in $Mbit/s$	100	100	11	11
Watchdog (no synchronization)	1	1	3	6
Lost packets	0	0	17	37
Wrong packet order	0	0	3	0

Table 1. Comparison of LAN and WLAN results.

in the wrong order. In each case, it was the previous packet, which was received right after the next one. In these cases the reflection of a sent packet was received.

A comparison of the different results for the two-party key exchange is shown in Table 1.

4 Conclusion and Future Work

The developed software simulation verifies the expectations and demonstrates that TPMs can be used in two different modes to establish the group key. These two proposed modes are the sequential and the parallel mode.

A protocol for a secure group key exchange using Tree Parity Machines and a first prototypical software implementation have been developed. Both are introduced and described in this work. Tree Parity Machines can be used to exchange a common secret group key in ad-hoc networks. No management facility or any explicit participants are needed for this exchange.

The prototype was tested in different network environments. The results show that the key exchange is performed quickly and the implementation is robust against failures. It is shown that the TPM key exchange is practical for the group key exchange without the need of any centralized facility taking care of the group management.

Further work has to be done to compare the TPM group key exchange protocol against others. This includes especially tests in a network simulator with significant more nodes and mobility scenarios. Bigger groups in WLAN might not be able to perform the key exchange in parallel mode because the bandwidth might be insufficient. In case that the parallel mode cannot be used, the linear mode must be taken. The disadvantage of this solution is the linear increase of time needed until a final group key is available; this time depends on the group size.

Another practical problem in WLANs is the requirement that all participants have to be within radio range of each other because no routing ability is implemented yet.

References

1. Challal, Y., Seba, H.: Group key management protocols: A novel taxonomy. International Journal of Information Technology **2** (2005) 105–118
2. Mattern, F.: Wireless future: Ubiquitous computing. In: Proceedings of Wireless Congress 2004, Munich, Germany. (2004)
3. Mattern, F., Sturm, P.: From distributed systems to ubiquitous computing - the state of the art, trends, and prospects of future networked systems. In Irmscher, K., Fähnrich, K.P., eds.: Proceedings of KIVS 2003, Springer-Verlag (2003) 3–25
4. Paar, C.: Pervasive computing and the future of crypto engineering. Invited talk at the École Polytechnic Fédéral de Lausanne (EPFL), Switzerland (2003)
5. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **22** (1976) 644–654
6. Kumar, S., Girimondo, M., Weimerskirch, A., Paar, C., Patel, A., Wander, A.S.: Embedded end-to-end wireless security with ecdh key exchange. In: Proceedings of the 46th IEEE Midwest Symposium On Circuits and Systems, Cairo, Egypt. (2003)
7. Anderson, R., Chan, H., Perrig, A.: Key infection: Smart trust for smart dust. In: Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04), Berlin, Germany. (2004) 206–215
8. Cagalj, M., Capkun, S., Hubaux, J.P.: Key agreement in peer-to-peer wireless networks. In: Proceedings of the IEEE (Special Issue on Cryptography and Security). (2005)
9. Kanter, I., Kinzel, W., Kanter, E.: Secure exchange of information by synchronisation of neural networks. Europhysics Letters **57** (2002) 141–147
10. Volkmer, M.: Entity authentication and authenticated key exchange with tree parity machines. Cryptology ePrint Archive Report 2006/112 (2006)
11. Fuhrmann, T., Di, P., Kutzner, K., Cramer, C.: Pushing chord into the underlay: Scalable routing for hybrid manets. Technical Report 2006-12, Fakultät für Informatik, Universität Karlsruhe (2006)
12. Perkins, C., Belding-Royer, E., Das, S.: Ad hoc on-demand distance vector (aodv) routing. Technical report, Network Working Group (2003)
13. Johnson, D.B., Maltz, D.A., Broch, J.: DSR The Dynamic source routing in ad hoc wireless networks. In: Ad Hoc Networking. Addison-Wesley (2001) 139–172
14. Volkmer, M., Wallner, S.: Tree parity machine rekeying architectures. IEEE Transactions on Computers **54** (2005) 421–427
15. Volkmer, M., Wallner, S.: A low-cost solution for frequent symmetric key exchange in ad-hoc networks. In: Proceedings of the 2nd German Workshop on Mobile Adhoc Networks, WMAN 2004. (2004) 128–137
16. Volkmer, M., Wallner, S.: Lightweight key exchange and stream cipher based solely on tree parity machines. In: Proceedings ECRYPT (European Network of Excellence for Cryptology) Workshop on RFID and Lightweight Crypto. (2005) 102–113