Tree Parity Machine Rekeying Architectures for Embedded Security

Markus Volkmer and Sebastian Wallner

Hamburg University of Technology
Department of Computer Engineering VI
D-21073 Hamburg, Germany
{markus.volkmer,wallner}@tu-harburg.de

Abstract. Nonclassical cryptographic technologies are considered in science and industry to provide alternative security solutions. They are motivated by the strong restrictions as they are often present in embedded security scenarios and in applications of pervasive computing. We investigate a low hardware-complexity cryptosystem for lightweight symmetric key exchange, based on two new Tree Parity Machine Rekeying Architectures (TPMRAs). The speed of a key exchange is basically only limited by the channel capacity. This work significantly improves and extends previously published results on TPMRAs. We evaluate characteristics of standard-cell ASIC design realizations as IP-core in 0.18μ-CMOS technology.

Keywords: Nonclassical cryptographic technologies, embedded security, pervasive computing, symmetric key exchange

1 Introduction

The investigation of alternative security primitives and nonclassical cryptographic technologies is stimulated by the strong restrictions present in ubiquitous and pervasive computing applications and the resource-limited devices in use. Handheld devices, smartcards, mobiles or other wireless communication devices require security concepts to be developed, in order have privacy and also commercial prosperity [1]. In sensor networks, RFID-systems or Near Field Communication (NFC), the devices in use as nodes of a network can impose severe size limitations and power consumption constraints. The available size for additional cryptographic hardware components is often limited [2, 3, 4]. To optimize a cost-performance-ratio regarding chip-area, channel bandwidth, power consumption and code-size with respect to a given platform (Microcontroller, FPGA, ASIC) represents a challenge [5].

Secure key exchange is considered most critical and complex in this context and of major importance with regard to security. Regarding applications in embedded systems, asymmetric (public-key) group-based cryptosystems based on Elliptic Curve Cryptography (ECC) (see e.g. [6]) and hardware-specific extensions for efficient arithmetic [7] are state-of-the-art. Without a reduction of the security, these representations allow to reduce the size of the numbers to calculate with. Yet, more complex expressions need to be calculated. The ring-based asymmetric cryptosystem NTRU [8, 9] also calculates on rather small numbers.

According to Paar [5] implementations of ECC on 16-bit microprocessors (clock-frequency ≤ 50 *MHz*) are feasible, while RSA and Diffie-Hellman are still hard. On an 8-bit microprocessor (clock-frequency ≤ 10 *MHz*) only symmetric algorithms are considered applicable given low data rates. Asymmetric algorithms here require an additional crypt-coprocessor. For the extreme case of an RFID-tag with around 1000 gates and no microprocessor available, only lightweight stream ciphers are considered applicable: ECC requires more than 10000 gates and DES alone already demands a few 1000 gates. Symmetric algorithms for this class are sought.

Although solutions for pervasive computing that allow to employ asymmetric cryptography are sought [10], the lack of infrastructure often present in this scenario penalizes approaches needing a central authority, like a trust center or a third trusted party, securely providing public keys. Threshold cryptography is still computationally intensive and a distributed certificate authority does not address the resource limitations of devices. After all, a frequent key exchange still remains of prohibitive cost, especially in the often changing topology of pervasive or ad-hoc networks. Cryptographic methods with appropriate computational efficiency, that also consider a certain message or protocol overhead, are inevitable. Seeking and investigating alternative approaches beyond efficient implementations thus remains a challenge for research. In practice, a necessary tradeoff between the level of security and the available resources or computation time often has to be faced.

We suggest to discuss a hardware solution for lightweight symmetric key exchange based on the synchronization of so-called *Tree Parity Machines* [11, 12]. We define two architecture-variants, using this key exchange concept and a trajectory mode, that allow fast successive key generation and exchange. It enables short key lifetimes through the achievable speed of a key exchange. We focus on a low hardware-complexity IP-Core solution for secure data exchange between resource-limited devices in pervasive computing. Feasible frequent rekeying and variable key lengths allow for flexible security levels especially in environments with moderate security concerns.

2 Key Exchange by Tree Parity Machines

The fast synchronization of two interacting identically structured Tree Parity Machines (TPMs) is proposed by Kinzel and Kanter [11, 12] as a method for symmetric key exchange. It does not involve large numbers and principles from number theory, however, Shamir et al. conferred to this interaction over multiple rounds as *a gradual type of Diffie-Hellman* [13]. Even more related, secret key agreement based on interaction over a public insecure channel is discussed under information theoretic aspects by Maurer and others [14, 15, 16, 17, 18, 19]. The exchange protocol is realized by an interactive adaptation (error-correction) 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. Each summation

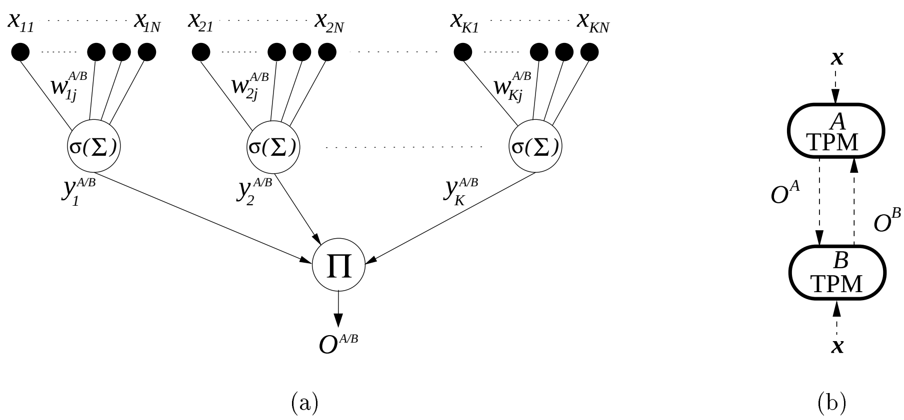


Fig. 1. (a) The Tree Parity Machine. A single output is calculated from the parity of the outputs of the summation units. (b) Outputs on commonly given inputs are exchanged between parties A and B for adaptation of their preliminary key.

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 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:

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

$\sigma()$ denotes the sign-function. The so-called *bit package* variant (cf. [11]) reduces transmissions of outputs by an order of magnitude down to a few packages. 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:

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

Coefficients are always bound to remain in the maximum range $[-L, L] \subseteq \mathbb{Z}$ by reflection onto the boundary values. Iterating the above procedure in as an interactive (error-correction) 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. 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 has never been communicated between the two parties and can be used to derive a common time-dependent final key by privacy amplification [18] or can be used directly. 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. There are $2^{KN}-1$ possible inputs in each iteration, yielding as many possible initializations for a pseudo random number generator.

A practical test for an accomplished key exchange is to test on successive equal outputs in a sufficiently large number of iterations, such that equal outputs by chance are excluded. Our investigations confirmed that the average synchronization time is distributed and peaked around 400 (i.e. thirteen 32-bit packages) for the parameters given in [11].

2.1 Trajectory Mode, Security and Attacks

Again consider that when the two parties are synchronous they also have the same outputs in each iteration. Communication can thus be stopped and each party then simply applies the adaptation (eq. 2) with its own output in order to have a next key from the trajectory in key-space. Using the *Trajectory Mode* this way avoids the security weakness as stated in [20], which assumes an ongoing communication. As soon as a new key is present, it is used for encryption. It can then be used block-wise or be used on a per-packet basis, depending on the concrete application. In any case, the key is only used to encrypt a certain small subset of the plaintext. This especially allows to realize short key lifetimes, enabling a certain security level by many smaller keys instead of one large key.

For the key exchange protocol without entity authentication, eavesdropping attacks have concurrently been proposed by Shamir et al. [13] and Kanter, Kinzel et al. [21, 22, 23]. The prevalent definition of a successful attack is having a 98 percent average overlap with the coefficients of one party, when parties A and B are already synchronous and thus successfully finished the key exchange and the communication. The overlap $|w^E(t) \cdot w^{A/B}(t)|$ is averaged over all summation units. The authors chose this definition, because of the strong fluctuations observed in the success probability. The attacks in [13, 21, 22, 23] can all be made arbitrarily costly and thus can practically be defeated by simply increasing the parameter L. The security increases proportional to L^2 while the probability of a successful attack decreases exponentially with L [21]. The approach is thus regarded computationally secure with respect to these attacks for sufficiently large L [24, 23].

The latest attack, which does not seem to be affected by an increase of L (but still by an increase of K) uses a hundred coordinated and communicating TPMs [23]. A successful attack according to the definition given above could be achieved with a probability of 0.5. The success probability of achieving a 99 percent average overlap drops down to 0.25. However, an attacker does not know, which of the $K \cdot N$ components of the coefficients (the key) are correct. In currently used symmetric encryption algorithms, the flipping of a single bit only already leads to a complete failure in decryption. Due to the only partial knowledge of an attacker on the final key, an added or included privacy amplification through hashing can further significantly decrease this knowledge and increase the secrecy of the final key (compare [16, 19, 18]) and also the security of the trajectory mode. It increases the entropy of the keys and destroys partial knowledge an attacker might have gained on the key from the known attacks.

Finally, note that all of the existing attacks refer to a non-authenticated key exchange, in which also man-in-the-middle attacks are possible. Given an appropriate mechanism for entity authentication and the application of a privacy amplification step, an appropriate level of security is provided at least for some applications in embedded security.

3 Tree Parity Machine Rekeying Architectures

Note that, with respect to a hardware implementation, only signs and bounded integers are processed within Tree Parity Machine key exchange. The result of the outer product in (Eq. 1) can be realized without multiplication. The product within the sum is only changing the sign of the coefficient. Thus, the most complex structure to be implemented is an adder and parallelism can be exploited. The branches are only based on a test for the sign or a test on equality to zero, also easily done in hardware. Furthermore, only sign-operations and additions are present in the adaptation (Eq. 2), well suited for a hardware implementation. The bit package exchange can either be realized serially or via a parallel bus, depending on the users requirements and the intended application. The amount of registers needed for storage increases in the bit package variant, finally imposing a tradeoff area versus speed. Equal pseudo-random inputs are realized by equally initialized Linear Feedback Shift Registers (LFSR). Different secret initial weights can be provided by an additional true random number generator. The synchronization criterion basically comprises a counter, that is increased when outputs are identical and reset at different outputs. Software experiments verified, that three identical bit-packages of 32 bit reliably indicate a successful synchronization for $L = 4$. The synchronization times are only determined by the properties of the algorithm and the capacity of the communication channel, as can be seen in Section 4.

The *Tree Parity Machine Rekeying Architectures* (TPMRAs) can be functionally separated into two main structures. One structure comprises the Handshake/Key Controller as well as the Bitpackage Unit and the Watchdog, the other structure contains the Tree Parity Machine Unit for calculating the basic TPM functions (Section 2). Figure 2 gives an overview of the hardware structure. The Handshake/Key Controller Unit handles the key transmission (eventually

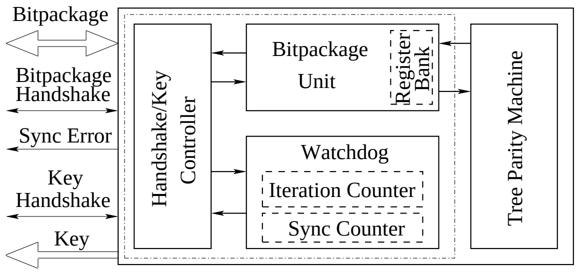


Fig. 2. Basic diagram of the Handshake/Key Controller with the Watchdog and the Bitpackage Unit. The Tree Parity Machine Unit may include a parallel or serial TPM computation structure.

after privacy amplification) with an encryption unit and the bit package exchange process with the other party by using a simple request and acknowledge handshake protocol. It approves the handling of different synchronization cycles between two key exchange parties in order to permit a regulated key- and bit package exchange process. A key is handed over when the synchronization process is finished, indicated by an acknowledge signal.

As described in Section 2, we implemented the bit package generalization of the protocol [11]. It reduces communication by an order of magnitude. The bit package exchange can then be realized serially or via a parallel bus, depending on the users requirements and the intended application environment. In both architectures, the Bitpackage Unit partitions the parity bits (Eq. 1) from the TPM Unit in tighter bit slices. In addition, it serializes the incoming bit packages from other TPM for the adaptation (Eq. 2). The Bitpackage Unit handles bit package lengths up to n bits (depending on the key length) for different parallel data exchange buses.

The Watchdog supervises the synchronization between the two parties, which is determined by the chosen parameters and the random initial values of the parties. The Iteration Counter in the Watchdog counts the number of exchanged parity bits. It generates a synchronization error (Sync Error), if there is no synchronization within a specific number of iterations. In this case, the synchronization process is triggered again. The Sync Counter is needed to determine the synchronization of the TPMs by comparing and counting equal output bit packages. It is increased when a sent bit package and the corresponding received bit package is identical and otherwise cleared. A synchronization is recognized when a specified number of equal bit packages is reached. Both Sync- and Iteration-Counter are programmable for variable average synchronization times subject to the chosen TPM structure. In the following, we describe details of the serial and parallel TPMRAs.

3.1 Serial Tree Parity Machine Unit

The serially realized TPM structure uses TDMA to calculate a parity bit and is a fully parameterizable hardware structure. The parameters K, N and L as well as the bit package length can be set arbitrarily in order to adopt this architecture variant for different system environments.

The serial TPM Unit consists of a TPM control state machine, a LFSR, a Weight Accumulator, a Parity Bit Computation and Weight Adjustment Unit and a memory (Figure 3).

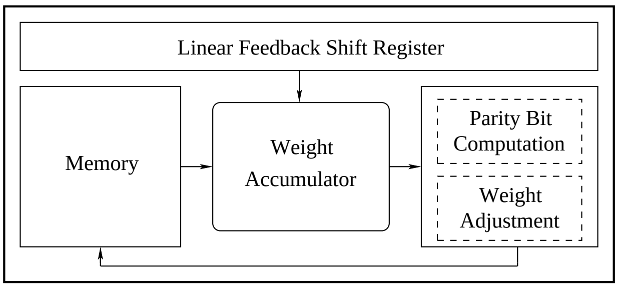


Fig. 3. The serial TPMRA structure. The TPM controller state machine is omitted for clarity.

The TPM controller is realized as simple finite state machine. It handles the initialization of the TPM, the adaptation with the parity bits of the bit package from the other party and controls the parity calculation and weight adjustment. The LFSR generates the pseudo random bits for the inputs $x_{kj}(t)$ of the TPM. The Parity Bit Computation computes the output parity (Eq. 1) and the Weight Adjustment Unit accomplishes the adaptation (Eq. 2). The Weight Accumulator computes each sum of the summation units. Each partial result must be temporarily stored in the memory, due to the serial processing of the summation units. The memory, implemented as a simple asynchronous S-RAM, stores the weights and the output bits from the summation units in order to process the bit packaging. It could also be implemented as a register file composed of several flip-flops. The memory size depends on the length of the key, which is equal to $K \cdot N \cdot L$. The number of cycles for calculating a n-bit package is $t_{BP} = (2n-1) \cdot (K \cdot N + K) + 3$.

3.2 Parallel Tree Parity Machine Unit

The parallel realization of the TPM Unit (Figure 4) has the same overall structure, but three parallel summation units are implemented and a register bank for each summation unit holds the weights as well as the parity bits of each unit.

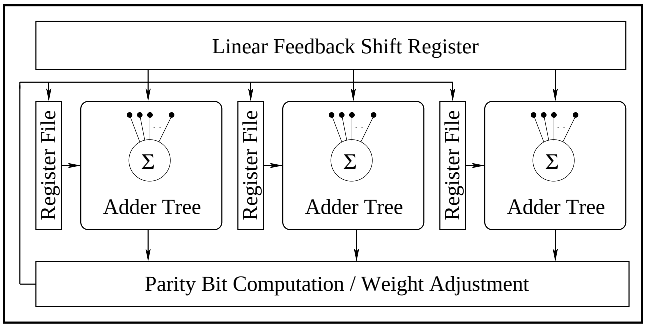


Fig. 4. The parallel TPMRA structure. The TPM controller state machine is omitted for clarity.

A summation unit consists of a pipelined adder tree designed to add $N = 11$ inputs. Each summation unit includes a $11 \cdot L$-bit register bank due to the need for parallel availability of data. In contrast to the serial TPM realization, the computation of parity and weight adjustment of each summation unit is also performed in parallel. The pipelined adder tree needs four clock cycles to calculate the summation with eleven inputs. In total, the whole architecture needs $t_{BP} = n+4$ cycles to compute an n-bit package including 4 pipeline cycles. It needs four clock cycles to compute a new key when using the trajectory mode explained in Section 2.1. The parallel TPM unit confines the parameterization due to the usage of the fixed adder tree. This results in a fixed number of summation units and a fixed number of inputs for each summation unit. The key length can only be varied via the parameter L, which does not influence the number of clock cycles. The parallel architecture has a fixed key length of $3 \cdot 11 \cdot 4 = 132$ bit at $L = 4$.

4 Implementation and Results

For the implementation of the TPMRAs in hardware, the choice of the key length as well as the choice for a serial or a parallel realization must be done with respect to the target environment, including the available silicon area, the available channel capacity and the timing.

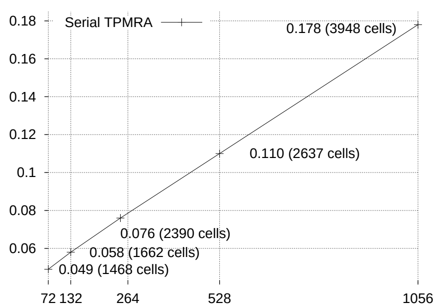
We designed and simulated a parameterizable serial TPMRAs and a fixed parallel TPMRA by using VHDL. While the FPGA-realization was used for easy prototyping, standard cell ASIC-realization prototypes were build to verify the suitability in typical embedded system components. The underlying process is a 0.18μ six-layer CMOS process with $1.8V$ supply voltage based on the UMC library [25]. The linear complexity of the protocol scales with the size $K \cdot N$ of the TPM structure, which defines the size $K \cdot N \cdot L$ of the key. We chose $K = 3$, a maximal $N = 88$ and $L = 4$ for the serial architecture. This leads to a key size of up to 1056 bit. The parallel TPM realization (with $K = 3$ and $N = 11$ fixed) has a key length of 132 bit with $L = 4$. The cell-area (Figure 5a) of the serial TPMRA scales approximately linear due to the linear increase in required memory and ranges around 0.11 square-millimeter for the investigated key sizes. The number in braces denotes used standard cells. Note, that most of the area is consumed by the memory, because of the necessary storage of the partial results (cf. Section 3). Yet, this influence is minor for an ASIC-realization, because here registers can be mapped more efficiently than on current FPGA architectures. The cell-area of the parallel TPMRA is 0.14 mm^2 (4559 cells).

The achievable clock-frequency (Figure 5b) ranges between 285 and 471 *MHz* for the investigated key lengths. It is significantly lower than in the parallel version, due to the memory access delay especially for larger key lengths. The parallel TPMRA can be clocked with 719 *MHz* (132 bit key).

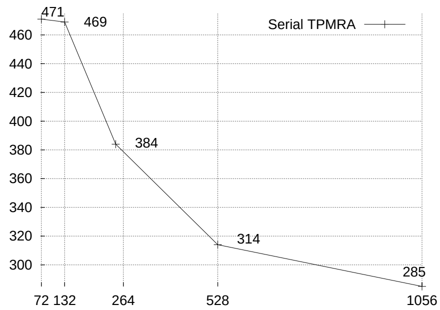
Additionally, we established the throughput (i.e. keys per second) subject to the average synchronization time of 400 iterations for different key lengths in Figure 5c. A practically finite channel capacity is neglected here. We assumed the maximally achievable clock frequency with regard to each key length, which can be achieved by Digital Phase Lock Loop (DPLL), regardless of the systems clock frequency. The serial TPMRA achieves a maximal theoretical throughput in the *kHz*-range. After the initial synchronization, the trajectory mode allows to increase the throughput by two orders of magnitude due to the reduced number of cycles for one bit package and the missing communication overhead. The parallel TPMRA yields a theoretical throughput of $20.5 \cdot 10^6$ keys per second.

Figure 5d shows throughput for real communication channels and their bandwidths. The chosen log-scale allows to still see the small difference regarding the throughput for Bluetooth, NFC and RFID in comparison to WLAN and PCI. For every protocol, we used the minimum available packet length due to our bit packages of 32 bit: PCI (burst mode) 32 bit, WLAN 801.11g 512 bit, Bluetooth 190 bit, NFC 64 bit and RIFD 94 bit. For the RFID channel we appointed a 10 *kbps* channel for simplicity.

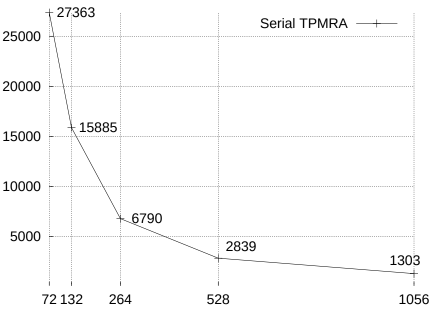
A comparison among the different communication channels indicates different slopes of the calculated throughput characteristics (Figure 5d). They denote the rising influence of the bit packaging calculations at smaller key lengths for channels of higher bandwidth such as WLAN (or PCI). Thus, the slope of the WLAN throughput characteristic is significantly higher than for Bluetooth, NFC



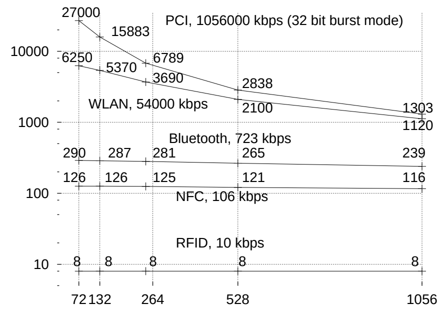
(a) Area [mm^2] vs. key length [bit]



(b) Speed [MHz] vs. key length [bit]



(c) Average key exchange rate [Hz] vs. key length [bit] (idealized infinite channel bandwidth)



(d) Average key exchange rate [Hz] log-scaled vs. key length [bit] (five real channels/protocols)

Fig. 5. Serial TPMRA post-synthesis area-optimized results vs. key length. Data refers to a UMC 0.18μ six-layer CMOS standard cell process.

and RFID. As expected, the influence of the channel bandwidth significantly determines the performance of the key exchange protocol. In the case of an 32 bit PCI-bus in burst mode (i.e. one bit package transfer per cycle), the theoretical maximum throughput (as in Figure 5c) can be achieved. Obviously, the bottleneck is the underlying communication-bus, as it is also typical in other domains (processor-bus-bottleneck).

5 Conclusions

We suggest to discuss a nonclassical cryptographic technology for low hardware-complexity lightweight authenticated symmetric key exchange, based on two variants of Tree Parity Machine Rekeying Architectures (TPMRAs). The silicon area ranges around 0.11 square-millimeter and the architectures allow to exchange keys of practical size up to the *kHz*-range. Efficient frequent rekeying and short key lifetimes can thus be implemented. Next to using sophisticated encryption algorithms like Rijndal (AES) or IDEA, for example, weak encryption algorithms (e.g. XOR or lightweight stream ciphers) may apply in moderate security scenarios as the security may (partly) rely on frequent rekeying.

We regard the TPMRAs as IP-cores for lightweight key exchange in embedded system environments. A particular focus can be smartcards or transponder-based applications such as RFID-systems, as well as devices in ad-hoc networks, in which a small area for cryptographic components is mandatory. They are especially suited for devices of limited resources and even more in moderate security scenarios.

A prototypical implementation of a TPMRA in a WLAN system was able to provide a new key for each 1500 byte package by immediate rekeying (no trajectory mode) and multiplexing the processes of rekeying and DES-encrypted communication. The prototypical implementation of the presented parallel TPMRA allows for a new 132 bit key for each block in a PCI-Bus burst mode access employing the trajectory mode. An experimental implementation into an RFID-system, a CAN-bus system for automotive applications and a WLAN ad-hoc-network are currently investigated.

A lightweight stream cipher variant, using the output bits directly was suggested already in [11] and its implementation in hardware is particularly suited for streaming applications. Key exchange between multiple parties is also possible and pursued in the project. This issue is relevant to secure group communication and multicast.

References

[1] Anderson, R.: Protecting embedded systems – the next ten years. In: Proc. of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2001. Volume 2162 of LNCS., Paris, France, Springer Verlag (2001) 1–2

[2] Stajano, F.: Security in pervasive computing. In: Proc. of the 1st International Conference on Security in Pervasive Computing (SPC 2003). Volume 2802 of LNCS., Springer Verlag (2003) 1

[3] Wollinger, T., Guajardo, J., Paar, C.: Cryptography in embedded systems: An overview. In: Proc. of the Embedded World 2003 Exhibition and Conference, Nürnberg, Germany, Design & Elektronik, Nürnberg (2003) 735–744

[4] Stanford, V.: Pervasive computing goes the last hundred feet with RFID systems. Pervasive Computing, IEEE Computer Science (2003) 9–14

[5] Paar, C.: Past and future of cryptographic engineering. Tutorial at HOT CHIPS 2003, Stanford University, USA (2003)

[6] Pelzl, J., Wollinger, T., Paar, C.: Low cost security: Explicit formulae for genus-4 hyperelliptic curves. In: 10th Annual Workshop on Selected Areas in Cryptography (SAC 2003), Springer Verlag (2003)

[7] Bailey, D., Paar, C.: Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. Journal of Cryptology **14** (2001)

[8] Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A ring-based public key cryptosystem. In Buhler, J., ed.: Proc. of Algorithmic Number Theory (ANTS III), Portland, Oregon. Lecture Notes in Computer Science, Springer-Verlag, Berlin (1998) 267–288

[9] Hoffstein, J., Silverman, J.: Optimizations for NTRU. In: Proc. of Public-Key Cryptography and Computational Number Theory, Warsaw. (2000)

[10] Hubaux, J.P., Buttyan, L., Capkun, S.: The quest for security in mobile ad hoc networks. In: Proceedings of ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC), Long Beach, CA (2001)

[11] Kanter, I., Kinzel, W., Kanter, E.: Secure exchange of information by synchronization of neural networks. Europhysics Letters **57** (2002) 141–147

[12] Klein, E., Mislovaty, R., Kanter, I., Ruttor, A., Kinzel, W.: Synchronization of neural networks by mutual learning and its application to cryptography. In: Proc. of Neural Information Processing Systems 2004 (NIPS 2004), Whistler, British Columbia, Canada (2004)

[13] Klimov, A., Mityagin, A., Shamir, A.: Analysis of neural cryptography. In: Proc. of AsiaCrypt 2002. Volume 2501 of LNCS., Queenstown, New Zealand, Springer Verlag (2002) 288–298

[14] Maurer, U.: Protocols for secret key agreement by public discussion based on common information. In: Advances in Cryptology – CRYPTO '92. Volume 740 of LNCS., Springer Verlag (1993) 461–470

[15] Maurer, U.: Secret key agreement by public discussion. IEEE Transactions on Information Theory **39** (1993) 733–742

[16] Brassard, G., Savail, L.: Secret-key reconciliation by public discussion. In: Advances in Cryptology – EUROCRYPT 1993. Volume 765 of LNCS., Springer-Verlag (1994) 410–423

[17] Wolf, S.: Strong security against active attacks in information-theoretic secret-key agreement. In: Advances in Cryptology — ASIACRYPT 1998. Volume 1514 of LNCS., Springer-Verlag (1998) 405–419

[18] Maurer, U., Wolf, S.: Secret key agreement over a non-authenticated channel — part iii: Privacy amplification. IEEE Transactions on Information Theory **49** (2003) 839–851

[19] Renner, R., Wolf, S.: Unconditional authenticity and privacy from an arbitrarily weak secret. In: Advances in Cryptology – CRYPTO 2003. Volume 2729 of LNCS., Springer-Verlag (2003) 78–95

[20] Kinzel, W., Kanter, I.: Interacting neural networks and cryptography. In Kramer, B., ed.: Advances in Solid State Physics. Volume 42. Springer Verlag (2002)

[21] Mislovaty, R., Perchenok, Y., Kanter, I., Kinzel, W.: Secure key-exchange protocol with an absence of injective functions. Phys. Rev. E **66** (2002)

[22] Kanter, I., Kinzel, W.: Neural cryptography. In: Proc. of the 9th International Conference on Neural Information Processing, Singapore (2002)

[23] Kanter, I., Kinzel, W., Shacham, L., Klein, E., Mislovaty, R.: Cooperating attackers in neural cryptography. Phys Rev. E **69** (2004)

[24] Rosen-Zvi, M., Klein, E., Kanter, I., Kinzel, W.: Mutual learning in a tree parity machine and its application to cryptography. Phys. Rev. E. **66** (2002)

[25] UMC: High Performance Standard Cells Design Kit Rev 2.2. (2001)