Tree Parity Machine Rekeying Architectures

Markus Volkmer and Sebastian Wallner

Manuscript received September 1, 2004; revised October 15, 2004.

Markus Volkmer and Sebastian Wallner are with Hamburg University of Science and Technology, Department of Computer Engineering VI, D-21073 Hamburg, Germany
Email: markus.volkmer@tuhh.de, wallner@tuhh.de

Abstract

The necessity to secure the communication between hardware components in embedded systems becomes increasingly important with regard to the secrecy of data and particularly its commercial use. We suggest a low-cost (i.e. small logic-area) solution for flexible security levels and short key lifetimes. The basis is an approach for symmetric key exchange using the synchronization of Tree Parity Machines. Fast successive key generation enables a key exchange within a few milliseconds, given realistic communication channels with a limited bandwidth. For demonstration we evaluate characteristics of a standard-cell ASIC design realization as IP-core in 0.18μ-technology.

Index Terms

K.4.4.f Security, K.6.5.a Authentication, B.7.1.b Algorithms implemented in hardware, C.3.h Ubiquitous computing, J.9.d Pervasive computing

I. INTRODUCTION

For embedded systems like handheld devices, smartcards, mobiles or other wireless communication devices security concepts need to be developed, in order to keep privacy and still (commercially) exploit the merits of such devices in widespread and everyday use [1]. This is, for example, of particular interest for the smartcard- or RFID-industry, where the secrecy of data is directly linked to the commercial prosperity of a product. Also, the economic importance to secure information technology applications in the automotive area is becoming eminent along with the protection of firmware, access control, anti-theft protection, up to scenarios like the hacking of vital vehicle functions such as an antilock braking system (see e.g. [2]).

Yet, the often relatively small size and severe power consumption constraints of these devices limit the available size for additional cryptographic hardware components [3], [4], [5]. This holds in particular for sensor networks, RFID-systems and near field communication devices. Secure hardware is thus especially demanded for ubiquitous and pervasive computing, and the need and research efforts manifest in first conferences on security in pervasive computing [6].

Hardware-cryptosystems are often based on hard-coded secret keys as the basic secret. It is good common practice to obey the often cited *Kerckhoffs Principle* [7] ('no security through obscurity') and not base the security of a crypto-system on the secrecy of the device or algorithm it employs. The security of a system is thus only as strong as the secrecy of the (fixed) keys.

But some of the most effective attacks on a crypto-system involve no ciphertext analysis but instead find flaws in the key-management. Furthermore, insecure bus communication as reported in [8] (regarding the video game console market), allows attacks still above the chip level by sniffing internal buses. In embedded system environments, functions are being realized (at least partly) in hardware and often lack online system access. The changing of a fixed key, as any other security update, is difficult or even impossible – i.e. too expensive.

The exchange of a common secret key over a public channel is dominated by methods based on number theory since the invention of the Diffie-Hellman key exchange protocol in 1976 [9]. Computational security is based on the difficulty of the discrete logarithm problem in *El Gamal* [10], which is considered as difficult as the factorization problem of a product of long prime numbers as in *RSA* [11]. Such asymmetric algorithms need to perform a lot of computational intensive arithmetics on typically limited embedded microcontrollers. In a particular GSM mobile phone, for example, two algorithms are combined to meet performance requirements: an asymmetrical algorithm with a 1024 bit key for key exchange and a symmetrical algorithm using only 128 bit for the key and voice encryption [12]. This also demonstrates the often necessary tradeoff between the level of security and the available resources.

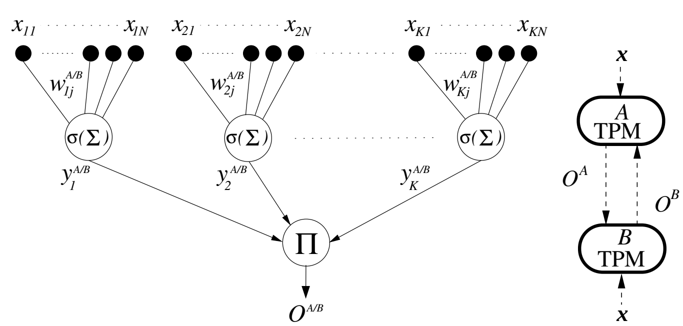
The state-of-the-art, regarding applications in embedded systems, is represented by *Elliptic Curve Cryptography* and the generalization to *Hyper-Elliptic Curves* (see e.g. [13]). Without a reduction of the security, these representations allow to reduce the size of the numbers to calculate with. Yet, more complex expressions need to be calculated. After all, a (frequent) key exchange is often of prohibitive cost, especially in the often changing topology of pervasive or ad-hoc networks.

In this paper we present a small hardware solution for secure data exchange with flexible security levels and short key lifetimes. It is based on a fast successive key generation and exchange process. We use a hardware-friendly algorithm for secure symmetric key exchange by synchronization of socalled *Tree Parity Machines* [14]. We define architectures, using this key exchange concept, that allow fast successive key generation and exchange. The key exchange ranges within milliseconds for realistic channels and can be performed in parallel (or multiplexed) to encryption and the encrypted communication process. Additionally, we provide the architectures with a flexible rekeying functionality to enable full exploitation of the achievable exchange rates. This particularly increases the cost for a successful immediate (online) attack, as opposed to a subsequent (offline) analysis on recorded information. Our focus is on secure data exchange between hardware components in embedded systems like RAM, FLASH-type ROM, (co-)processors and on bus-communication in general. Environments in which security can also be of moderate concern are also considered.

In the following, we introduce the neural network structure and a learning algorithm (section II), also in order to already point out advantageous properties for a hardware realization. The synchronization effect leading to the key exchange property is explained. Algorithmic security implications on the realization of our architecture are described in section III. Section IV comprises the architectural design of the proposed hardware component with its rekeying functionality. Here, we also refer to design decisions prepared in the previous section. In section V, we present results from an FPGA and an ASIC implementation on silicon area, possible clock and key exchange rates (throughput). We conclude the paper in section VI with a short summary and an outlook on possible further extensions also referring to current research activities.

II. Tree Parity Machine Key Exchange

In [14], Kanter et. al. proposed a symmetric key exchange method based on the fast synchronization of two identically structured Tree Parity Machines (TPMs). The particular tree structure has non-overlapping binary inputs, discrete weights and a single binary output as depicted in Fig. 1a.



(a) (b)

Fig. 1. (a) The tree parity machine (TPM) generates a single output – the parity of the outputs of the hidden units. (b) For mutual learning, outputs on commonly given inputs are exchanged between the two parties A and B.

Studying interacting neural networks in general (cf. [15], [16], [17]), the authors focused the phenomenom of fast synchronization by mutual learning TPMs and its potential for a cryptographic approach, not involving large numbers and methods from number theory. Their exchange protocol is realized implicitly by a mutual adaptation process between the two parties A and B, not involving large numbers and methods from number theory.

A. Structure of a Tree Parity Machine

In the following, we describe the implemented parallel-weights version using hebbian learning (cf. [14]). Weights are identical in both TPMs after synchronization. The anti-parallel-weights version, using anti-hebbian learning and leading to inverted weights at the other party, is omitted for brevity. The notation A/B denotes equivalent operations for the parties A and B. A single A or B denotes an operation which is specific to one of the parties.

The TPM (see Fig. 1a) consists of K hidden units ($1 \leq k \leq K$) in a single hidden-layer with non-overlapping inputs (the tree structure) and a single unit in the output-layer.

Each hidden 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 $O^{A/B}(t) \in \{-1, 1\}$, given bounded weights $w_{kj}^{A/B}(t) \in [-L, L] \subseteq \mathbb{Z}$ (from input unit j to hidden unit k) and common random inputs $x_{kj}(t) \in \{-1, 1\}$, is calculated by a parity function of the signs of summations:

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

The common random inputs can also be kept secret between the parties, yielding authentication (see Section II-B). σ is a party-specific modified sign-function, that defines an agreement between the two parties on an opposite sign in case of a sum $\alpha_k^{A/B}(t) \in \mathbb{Z}$ of zero:

$$
\sigma(\alpha_k^{A/B}(t)) := \begin{cases} 1 & , \alpha_k^{A/B}(t) > 0 \ \vee \ \alpha_k^{A}(t) = 0 \\ -1 & , \alpha_k^{A/B}(t) < 0 \ \vee \ \alpha_k^{B}(t) = 0 \ . \end{cases}
\tag{2}
$$

From the communicated output, the outputs of the hidden units cannot be uniquely determined. There are multiple combinations for a signed or unsigned output, depending on the number of hidden units K.

B. Key Exchange by Mutual Learning and Synchronization

The so-called *bit package* variant was chosen for implementation (cf. [14]). Due to an reduction of (physical) output exchanges by an order of magnitude, it is advantageous for practical communication channels with a certain protocol overhead.

Parties A and B start with an individual randomly generated initial weight vector $w_{kj}^{A/B}(t_0)$ – their secret. 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 Fig. 1b). The b sequences of hidden states $y_k^{A/B}(t) \in \{-1, 1\}$ are stored for the subsequent learning process.

A hebbian learning rule is applied to adapt the weights, using the b outputs and b sequences of hidden states. They are changed only on an agreement $O^A(t) = O^B(t)$ on the parties' outputs. Furthermore, only weights of those hidden units are changed that agree with this output, i.e. if $O^{A/B}(t) = y_k^{A/B}(t)$:

$$w_{kj}^{A/B}(t) := w_{kj}^{A/B}(t-1) + O^{A/B}(t)\ x_{kj}(t)\ . \tag{3}$$

Updated weights are bound to stay in the maximum range $[-L, L] \subseteq \mathbb{Z}$ by reflection onto the boundary values

$$w_{kj}^{A/B}(t) := \begin{cases} \text{sign}\Big(w_{kj}^{A/B}(t)\Big)\ L & , |w_{kj}^{A/B}(t)| > L \\ w_{kj}^{A/B}(t) & , \text{otherwise.} \end{cases} \tag{4}$$

In iterating the above procedure, each component of the weight vectors performs a random walk with reflecting boundaries. This implies a trajectory in a weight space of $(2L+1)^{KN}$ points. 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 (4), the distance between the components is successively reduced to zero. Synchrony is achieved when both parties have learned to produce each others outputs. They remain synchronized (see learning rule (3)) and continue to produce the same outputs on every commonly given input. This effect in particular leads to common weight-vectors in both TPMs in each of the following iterations. These weights have never been communicated between the two parties and can be used as a common time-dependent key for encryption and decryption respectively. Such secret key agreement based on interaction over a public insecure channel is also discussed under information theoretic aspects by Maurer [18], especially with regard to unconditional security. Furthermore, synchrony is achieved only for *common* inputs. Thus, keeping the common inputs secret between A and B can be used to have an authenticated key exchange. There are $2^{KN} - 1$ possible inputs in each iteration, yielding as many possible initializations for a pseudo random number generator. Shamir et al. conferred to such a synchronization over multiple rounds as *a gradual type of Diffie-Hellman* key exchange [19], because Diffie-Hellman has a single round that transmits several bits. Obviously, a test for synchrony cannot practically be defined by checking whether weights in both nets have become identical. One therefore tests on successive equal outputs in a sufficiently large number of iterations t_{min}, such that equal outputs by chance are excluded:

$$\forall t \in [t', \cdots, t' + t_{min}] : \; O^A(t) = O^B(t) \; . \tag{5}$$

The number of outputs (bits) required to achieve synchronization is lower than the size of the key [20]. Synchronization time is finite for discrete weights. It is almost independent on N and scales with $\ln N$ for very large N. Furthermore, it is proportional to L^2 [20]. Our investigations/experiments confirmed that the average synchronization time is distributed and peaked around 400 for the parameters given in [14].

III. Security and Rekeying Functionality

The symmetric key-exchange protocol can generate long keys by fast calculations and building the secure channel is of linear complexity. It scales with the size $K \cdot N$ of the TPM structure [14], which defines the size $K \cdot N \cdot L$ of the key. In order to still allow comparisons with the literature we chose $L = 4$ for our implementation. The time to synchronize roughly doubles in comparison to $L = 3$, while for the attacker the same time increases by orders of magnitude (see [21]).

The security of the key exchange manifests in algorithm-specific properties and can be fully exploited by appropriate hardware design. Next to other general (algorithmic) aspects on the security of the exchange method as described in [14], [19] , the tracking of the weights is hard in comparison to synchronization – practically even harder when implemented in hardware.

The key exchange protocol has been attacked by several eavesdropping approaches, which always require full knowledge of the TPM structure in use. We will describe their basic properties in order to clarify the security aspect. Due to the nature of the key exchange and its attacks, only probabilistic definitions of a 'successful attack' can be provided. One can distinguish between two classes of attacks. The first class comprises attacks, that can be defeated by appropriately increasing the parameter L. The consequence is, that the learning time of an attacker is significantly longer than the synchronization time. The security increases proportional to L^2 while the probability of a successful attack decreases exponentially with L [20]. Among these so defeatable attacks, which try to synchronize faster than the two parties [22], are the *Naive Attack*, that uses a single or an ensemble of several identically structured TPMs. The *Genetic Attack* even comprises a population of thousands of TPMs, whose internal representations are optimized by a genetic algorithm [19]. A successful attack is defined here as synchronizing faster than the parties A and B and could be realized for $K = 2$ in 50% of all cases. But, already for $L = 3$, this attack has shown to be less effective than the *Flipping Attack*. The complexity of such attacks (especially in hardware), with hundreds or thousands of TPMs plus an additional (genetic) algorithm, is obviously high. The Flipping Attack defines a successful attack as having 98% overlap with the weights of one party, when parties A and B are already synchronous. For $L > 2$, an ensemble of 10000 Flipping Attackers was found less effective than a single attacker, which revokes its practical use [22].

All of the previously sketched attacks can be made arbitrarily costly and thus practically defeated by increasing L, which significantly decreases the probability of a successful attack. The approach thus remains computationally secure for sufficiently large L [21], [23].

The only attack, which does not seem to be affected by an increase of L (but still by an increase of K) is the socalled *Majority Flipping Attack*. It uses a hundred of coordinated and communicating TPMs [23]. Yet, the given definition of a successful attack is problematic: When A and B have fully synchronized, the attacker has 98$\%$ average overlap (i.e. a fraction) with the weights, in 50$\%$ of all cases. For 99$\%$ average overlap, the probability reduces to 25$\%$. This indicates that the difficulty lies in achieving the last percents. The authors chose this definition, because of the strong fluctuations they observed in the success probability. But the definition of overlap is an average overlap over all hidden units. Thus an attacker does not know, which of the $K \cdot N$ components of the weights (the key) are correct in a real attack scenario. In currently used symmetric encryption algorithms, the flipping of a single bit only already leads to a complete failure in decryption. Thus practically, one still has to perform a subsequent brute-force attack, which would then only be successful in 50% of all cases. Keep in mind, that A and B already have one key (while the attacker has 98% with a probability of 0.5) and start encryption and data transmission. The attacker needs to perform his brute force attack plus the attack on the encrypted data in parallel. Furthermore, the rekeying principle and the achievable short key lifetimes (cf. Section III-A) aim at an online usage of the exchanged keys for secure transmission.

All formulated attacks can hardly be performed online. Only an offline attack on the previously recorded exchanged information seems realistic. Last but not least, note that all of the existing attacks are based on knowing the common inputs and thus refer to a non-authenticated key exchange, in which man-in-the-middle attacks are possible as well.

A. Feasible Immediate Rekeying

We propose to minimize the key lifetime as much as possible employing *immediate rekeying*, that allows to exploit the speed of key exchange and features of our hardware component. Such a rekeying process normally is to be avoided due to the computational cost of a new key exchange. Strategies are developed to increase the key lifetime without affecting the security (see e.g. [24]). Yet, using the TPM principle allows for efficient rekeying in the *kHz*-range (see also Section V).

Next to several other propositions (cf. [14]) concerning the en-/decryption, one particular proposition (cf. [16]) is to take each (common) weight vector after synchronization for en-/decryption. On the one hand, a new potential key is present in each step, which can then be used block-wise. On the other hand, an opponent then also has the chance to synchronize using the ongoing communication (cf. Section II-B) and get a key.

We suggest to permanently generate (i.e. synchronize) new keys in parallel or multiplexed to the encryption-transmission-decryption of data, using the most recent key exchanged. In this case, the key is only used to encrypt a certain small subset of the plaintext. As soon as a new key has been exchanged, it is used for encryption. This especially allows to realize short key lifetimes, enabling a certain security level by many smaller keys instead of one large key.

Consequently, in our hardware design, we allow an external unit to demand a key exchange service. Our TPMRAs will continuously synchronize new keys, as long as data needs to be exchanged. Once a crypt-unit uses the first key, synchronization is triggered again to always provide a next key. In this way, the related hardware resources are consequently used and keys are exchanged at a maximum rate subject to hardware constraints, average synchronization time and available channel bandwidth. Furthermore, it allows to implement services like periodic or even adaptive rekeying. Security is thus increased in our hardware implementation through feasible immediate rekeying, the mere speed of key exchange and the achievable short key lifetimes.

IV. Tree Parity Machine Rekeying Architectures

It is important to note that, with respect to a hardware implementation, only signs and bounded integers are processed within the algorithm. The result of the outer product in (1) can be realized without multiplication. The product within the sum is only changing the sign of the weight. Thus, the most complex structure to be implemented is an adder. The complexity of such a unit is thus even less than the complexity of a linear filter, which requires a full multiply-accumulate structure. Yet, the inherent parallelism can be exploited here as well. The branches in (2) 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 learning rule (3), 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 vs. speed.

Equal (pseudo-)random inputs are realized by equally initialized Linear Feedback Shift Registers (LFSR) or a Cyclic Redundancy Code (CRC). Different (secret) initial weights can either be fixed (device-specific), or they can be provided by an additional application-specific device or by a thermal noise device. The synchronization criterion (5) basically comprises a counter, that is increased when outputs are identical and resetted at different outputs. Software experiments verified, that 96 identical outputs (i.e. 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 V.

The proposed Tree Parity Machine Rekeying Architectures (TPMRAs) are functionally separated into two main structures. One structure essentially comprises the Key Handshake and Bit Package Control. The other structure contains the TPM Unit and its control state machine.

A. Key Handshake and Bit Package Control

As described in Section II-B, we implemented the *bit package* generalization of the protocol (cf. [14]). The overall structure of the TPMRAs is shown in Fig. 2. It consists of three functional blocks: a Key Handshake and Bit Package Control, the TPM unit and a Watchdog timer.

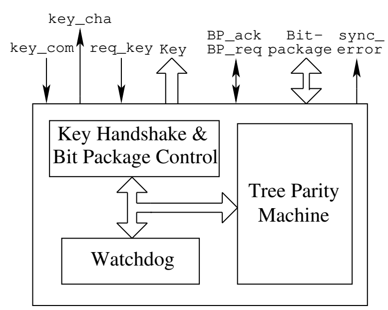


Fig. 2. Basic diagram and functional blocks of the Tree Parity Machine Rekeying Architectures.

The Watchdog timer supervises the number of interactions needed for a key-exchange between two parties (5). If there is no synchronization within a specific time (remember that the synchronization time is distributed), a signal (`sync_error`) indicates a synchronization error. It is programmable for variable average synchronization times subject to the chosen TPM structure.

The Key Handshake and Bit Package Control handles the key transmission with an encryption unit and the bit package exchange process with the other party. It accomplishes the bit packaging by partitioning the parity bits from the TPM unit in tighter bit slices. Due to different computation cycles between two key exchange parties, the rekeying procedure employs a key request (`req_key`), a key changed (`key_cha`) and a key commit (`key_com`) handshake protocol (see Fig. 2). A key is handed over via the internal bus (`Key`) to an encryption unit when the synchronization process is finished. For our application domain in embedded system environments, we choose a fixed bit package length of 32 bit for physically parallel exchange and synchronization over a 32 bit wide bus (`Bit Package`). The bit package exchange process uses a simple request/acknowledge handshake protocol (`BP_ack`, `BP_req`).

B. Tree Parity Machine Unit

The TPM unit comprises the logic for the TPM structure, such as the logic for calculating the parity bits as explained in Section II. It consists of the TPM control, a Cyclic Redundancy Code (CRC) generator, a Parity Computation unit and a Weight Adjustment unit. A register bank holds the data for the hidden unit and the weights of the network as shown in Fig. 3.

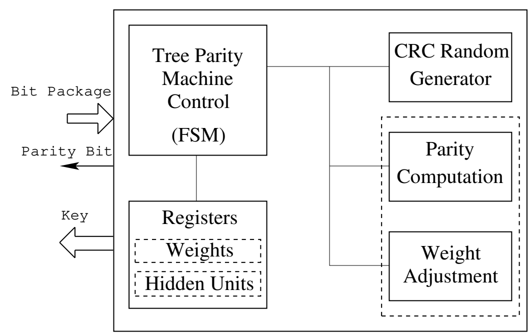


Fig. 3. Internal structure of the Tree Parity Machine Unit.

The TPM control is realized as simple finite state machine (FSM) which executes the initialization of the TPM and the learning process with the bit package from the other party. The Parity Computation unit calculates the summation and the parity bit (1) and 2). The weight adjustment unit accomplishes the learning rule (3) and (4).

The CRC random generator generates the pseudo random bits for the inputs of the TPM. It is initialized by a vector which is equal for both parties. For the purpose of authentication, the initial value would have to be kept secret.

V. Implementation and Performance

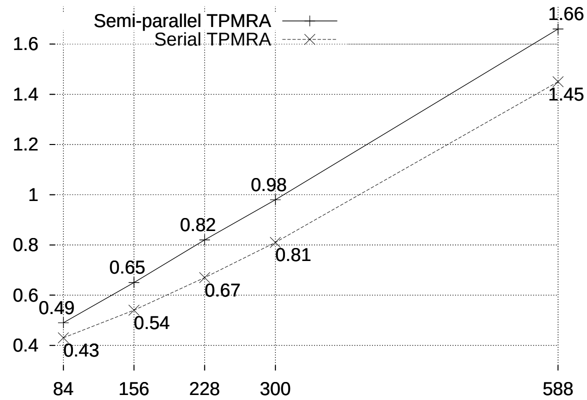
We designed and simulated parameterizable, serial and semi-parallel TPM Rekeying Architectures, using VHDL to implement a FPGA- and an ASIC-realization. For both architectures we appoint the integer range L to 4, as explained in Section III. In the serial architecture, the synaptic summation is performed by Time Devision Multiple Access (TDMA) of an L-bit adder, while the semi-parallel form uses TDMA of six L-bit adders in parallel. The details of the TPMRA implementations (key length $K \cdot N \cdot L$, serial or semi/fully-parallel realization) must be chosen with respect to the target environment, including the used parameters, the timing, the available channel capacity and the available chip-area, of course.

We realized a two party prototype system based on two *XCV300E-8 (Virtex E)* FPGAs from *Xilinx* in order to investigate and demonstrate the functionality of our architectures. Standard cell ASIC prototype realizations were built to verify the suitability of the TPMRAs for typical embedded system components. We chose $K = 3$ and varied N up to 49 for a resulting key size of $3 \cdot 49 \cdot 4 = 588$ bit. This choice for N already allows a remarkable key length and still keeps the average synchronization time low (cf. [14]). The underlying process was a 0.18μ six-layer CMOS process with $1.8V$ supply voltage based on the *UMC library*. The design was synthesized using the *Synopsys Design-Compiler* and was mapped using *Cadence Silicon Ensemble*.

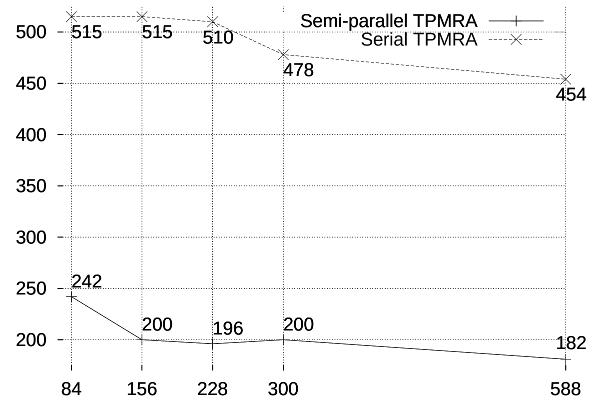
The area (Fig. 4a) of the TPMRA realizations scale approximately linear (around one square-millimeter) due to the linear complexity of the adders. The serial TPM realization consumes less area (i.e. less hardware resources). Note, that most of the area is consumed by the bit packaging, because of the necessary storage of the inputs for the learning (cf. Section II-B). Yet, this influence is minor for the ASIC-realization, because here registers are more efficiently mapped than on current FPGA architectures.

Obviously, the achievable clock speed (Fig. 4b) in the serial variant is significantly higher than in the semi-parallel version. This is due to the necessity of a longer clock tree for the additional registers to store partial results.

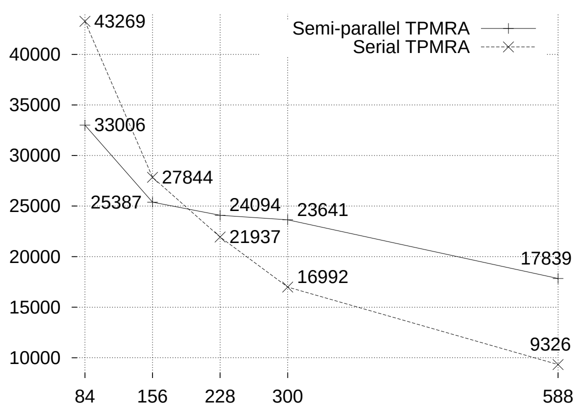
Additionally, we established the throughput (i.e. keys per second) subject to the average synchronization time of 400 iterations for different key lengths in Fig. 4c. We assumed the maximally achievable clock frequency with regard to each key length, which can be achieved by Digital Phase Lock Loop (DPLL), regardless of the systems clock frequency. Furthermore, we appointed the average synchronization time of 400 iterations for all key lengths, although it is really always less than the size of the key (a worst-case scenario). This data refers to an idealized infinite channel bandwidth, neglecting the transmission delay. For key lengths smaller than approximately 180 bit, the serial TPMRA has a higher throughput (in the range of $2.5 \cdot 10^4$ to more than $4 \cdot 10^4$ keys per second) due to the higher clock frequency (Fig. 4b). Beyond this



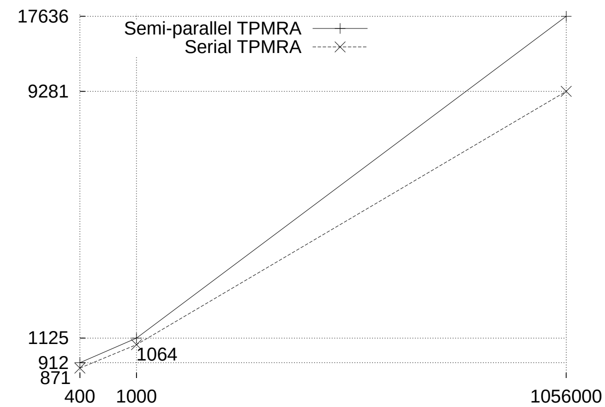
(a) Area $[mm^2]$ vs. key length $[bit]$



(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 $[Hz]$ vs. the channel bandwidths $[kbps]$ of I^2C, CAN and PCI (burst mode)

Fig. 4. Post-synthesis results for chip-area (logic) (a) and achievable clock-frequency (b) vs. key length. Average key exchange rate (avg. synchronization time of 400 iterations) vs. key length is plotted in (c). A practically finite channel capacity is neglected here. Plot (d) is log-scaled and shows average key exchange rates for a 588 bit key and a selection of typical channels with their capacities. All data refers to a UMC 0.18 micron six-layer standard cell process.

point, the semi-parallel version achieves a higher throughput, exploiting the parallel computation.

Figure 4d shows the same information, but for three real communication channels and their bandwidths, given a key length of 588 bit. The chosen log-scale allows to see the small difference regarding the throughput (up to around 1000 keys per second) for an I^2C and CAN-

bus. Only for buses of higher bandwidth such as the PCI-bus, the two architectures show a significantly different throughput (reaching the kHz-range). In the case of an 32 bit PCI-bus in burst mode, the theoretical maximum throughput (as in Fig. 4c) can be achieved. We also considered other bus systems (e.g. packet based systems like WLAN). The results are similar, due to their small bandwidth in comparison to the PCI-bus. Obviously, the bottleneck is the underlying communication-bus, as it is also typical in other domains (processor-bus-bottleneck). The concrete implementations of the components of the TPMRA influence the time necessary for the calculation of one output bit and the time for the assembly of a bit package. Yet, with the communication being the bottleneck, to optimize the TPMRA with regard to area is preferable. Given a high-speed communication channel, the proposed key exchange and rekeying in the kHz-range allows us to use rather weak encryption algorithms (cf. Section III), as the security may rely on fast rekeying. Of course, any other more sophisticated encryption algorithm like AES or 3-DES can also be used.

The achievable average key-exchange rates of the TPMRAs in the kHz-range, allow to increase the security through a feasible frequent key exchange. Short key lifetimes can be realized efficiently. Also, any successful online attack must at least achieve the same performance, requiring significant hardware expenses. This does not appear to be feasible. Using different keys for encryption and transmission of different blocks of data, increases the difficulty for an attack on the encrypted data.

Due to the small area in the range of one square-millimeter, we regard the field of application principally as an IP-core in embedded system environments. A particular focus can be smartcards or transponder-based applications such as RFID-systems and devices in ad-hoc networks [25], in which a small area for cryptographic components is mandatory.

VI. Summary and Outlook

We presented a solution for secure communication in embedded system environments via Tree Parity Machine Rekeying Architectures. Our investigations confirm the results as presented in [14] and stress the advantages of a hardware implementation. The silicon area lies within a square-millimeter and allows to exchange keys of practical size within about a millisecond. The proposed exchange in parallel to encryption-transmission-decryption also allows for efficient rekeying schemes and short key lifetimes.

Next to algorithmic extensions to further increase the security [26], [27], [28], [29], architectural improvements or variants include a fully serial realization with TDMA usage of a single TPM unit. This further decreases the area consumption but at the cost of an increase in necessary cycles for one output bit. A stream cipher variant, using output bits directly via the *Blum-Blum-Shub* bit generator, was suggested already in [14] and its implementation in hardware is particularly suited for streaming applications. The relatively small size of the TPMRAs allows an implementation in embedded systems with only small overhead. They are especially suited for devices of limited resources and even more in moderate security scenarios. Consequently, the integration of our architectures into such a system and its practical evaluation is subject to future work.

ACKNOWLEDGMENT

The project has been motivated by the fundamental research of Ido Kanter (Department of Physics, Bar-Ilan University, Israel) and Wolfgang Kinzel (Institute for Theoretical Physics III – Computational Physics, University of Würzburg). We are especially grateful to Wolfgang Kinzel and Andreas Ruttor for the inspiring discussions in Würzburg.

The authors would like to thank project collaborators Sascha Mühlbach for VHDL-Coding and André Schaumburg for performing software simulations and extensive testing. We also thank Karl-Heinz Zimmermann and Rawat Siripokapirom (Computer Engineering VI, Hamburg University of Science and Technology) for helpful discussions.

We acknowledge the support by Andreas Mäder of University of Hamburg, allowing and assisting us to use their industry-standard synthesis-toolchain for our ASIC design.

REFERENCES

[1] R. Anderson, "Protecting embedded systems – the next ten years (invited talk)," in *Proc. of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2001*, ser. LNCS, vol. 2162. Paris, France: Springer Verlag, May 14-16, 2001, pp. 1–2.

[2] C. Paar, "Eingebettete Sicherheit im Automobil. ESCAR – Embedded IT-Security in Cars, Cologne, Germany," 18.-19. Nov. 2003.

[3] E. Bovelander, "Smart card security," in *State of the Art in Applied Cryptography – Course on Computer Security and Industrial Cryptography (Revised lectures)*, ser. LNCS, B. Preneel and V. Rijmen, Eds. Springer Verlag, Jun 1997, vol. 1528.

[4] K. Vedder and F. Weikmann, "Smart cards – requirements, properties, and applications," in *State of the Art in Applied Cryptography – Course on Computer Security and Industrial Cryptography (Revised lectures)*, ser. LNCS. Springer Verlag, June 1997, vol. 1528, pp. 307–331.

[5] T. Wollinger, J. Guajardo, and C. Paar, "Cryptography in embedded systems: An overview (invited paper)," in *Proc. of the Embedded World 2003 Exhibition and Conference*. Nürnberg, Germany: Design & Elektronik, Nürnberg, Feb. 18-20 2003, pp. 735–744.

[6] F. Stajano, "Security in pervasive computing (invited talk)," in *Proc. of the 1st International Conference on Security in Pervasive Computing (SPC 2003)*, ser. LNCS, vol. 2802. Springer Verlag, 2003, p. 1.

[7] A. Kerckhoffs, "La cryptographie militaire," *Journal des sciences militaires*, vol. IX, pp. 5–38, Jan. 1883, (pp. 161-191, Feb. 1883).

[8] A. Huang, "Keeping secrets in hardware: the Microsoft X-BOX™case study," in *Proc. of the Workshop on Cryptographic Hardware and Embedded Systems 2002 (CHES 2002)*, ser. LNCS, vol. 2523. Springer Verlag, August 13 - 15 2003, pp. 213–227.

[9] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, November 1976.

[10] T. E. Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. IT-31, no. 4, pp. 469–472, 1985.

[11] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, feb 1978.

[12] R. . Schwarz, *TopSec GSM, Data sheet PD0757.6904.21*.

[13] J. Pelzl, T. Wollinger, and C. Paar, "Low cost security: Explicit formulae for genus-4 hyperelliptic curves," in *Tenth Annual Workshop on Selected Areas in Cryptography (SAC 2003)*. Springer Verlag, 2003.

[14] I. Kanter, W. Kinzel, and E. Kanter, "Secure exchange of information by synchronization of neural networks," *Europhysics Letters*, vol. 57, no. 1, pp. 141–147, 2002.

[15] R. Metzler, W. Kinzel, and I. Kanter, "Interacting neural networks," *Phys. Rev. E*, vol. 62, no. 2, pp. 2555–2565, 2000.

[16] W. Kinzel and I. Kanter, "Interacting neural networks and cryptography," in *Advances in Solid State Physics*, B. Kramer, Ed. Springer Verlag, 2002, vol. 42.

[17] S. Bornholdt and H. Schuster, Eds., *Handbook of Graphs and Networks*. Wiley VCH, 2003, ch. Theory of interacting neural networks.

[18] U. Maurer, "Secret key agreement by public discussion," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 733–742, 1993.

[19] A. Klimov, A. Mityagin, and A. Shamir, "Analysis of neural cryptography," in *Proc. of AsiaCrypt 2002*, ser. LNCS, vol. 2501. Springer Verlag, 2002, pp. 288–298.

[20] R. Mislovaty, Y. Perchenok, I. Kanter, and W. Kinzel, "Secure key-exchange protocol with an absence of injective functions," *Phys. Rev. E*, vol. 66, no. 066102, 2002.

[21] M. Rosen-Zvi, E. Klein, I. Kanter, and W. Kinzel, "Mutual learning in a tree parity machine and its application to cryptography," *Phys. Rev. E.*, vol. 66, no. 066135, 2002.

[22] I. Kanter and W. Kinzel, "Neural cryptography," in *Proc. of the 9th International Conference on Neural Information Processing*, 2002.

[23] L. N. Shacham, E. Klein, R. Mislovaty, I. Kanter, and W. Kinzel, "Cooperating attackers in neural cryptography," dec 2003, preprint www.arxiv.org/cond-mat/0312068.

[24] M. Abdalla and M. Bellare, "Increasing the lifetime of a key: A comparative analysis of the security of rekeying techniques," in *Advances in Cryptology – Asiacrypt 2000 Proceedings*, ser. LNCS, T. Okamoto, Ed., vol. 1976. Springer Verlag, 2000.

[25] W. Rankl and W. Effing, *Smart Card Handbook*. Wiley & Sons Ltd., 2000.

[26] M. Rosen-Zvi, I. Kanter, and W. Kinzel, "Cryptography based on neural networks – analytical results," *J. Phys. A: Math. Gen.*, vol. 35, no. 47, pp. L707–L713, 2002.

[27] A. Ruttor and W. Kinzel, "Repulsive feedback mechanisms in neural cryptography," 2003.

[28] R. Mislovaty, E. Klein, I. Kanter, and W. Kinzel, "Public channel cryptography by synchronization of neural networks and chaotic maps," *Phys. Rev. Lett.*, vol. 91, no. 118701, 2003.

[29] A. Ruttor, W. Kinzel, L. Shacham, and I. Kanter, "Neural cryptography with feedback," *Phys. Rev. E*, vol. 69, 2004.

Markus Volkmer worked in the areas of Fuzzy Image Processing, Agent-Systems and Object Recognition in the Computer Engineering Group of the University of Bielefeld, Germany. He received the degree *Diplom-Informatiker* in Computer Science from the University of Bielefeld in 1998. Recently, he has been a Research Associate at the Department of Computer Engineering VI of the Hamburg University of Science and Technology (TUHH), Germany, and is about to finish his PhD-thesis. His research interests include Signal Processing, Artificial Neural Networks, Embedded Security and Security in Ad-hoc Networks. His current focus is on effects of interneuron- and internetwork-coupling and -synchronization and their utility in information engineering applications. Markus Volkmer is a member of the Gesellschaft für Informatik e. V. (GI).

Sebastian Wallner received the Dipl.-Inform. (MSc) degree in Computer Science from the University of Hamburg, Germany, in 2000. In 2000, he joined the department of Computer Engineering VI of the Hamburg University of Science and Technology (TUHH) as a Research Associate where he is currently pursuing his PhD. His research interests include reconfigurable computing, heterogeneous computing systems, parallel computing architectures and cryptographic hardware design. His work on a Configurable System-on-Chip Architecture for Embedded Devices was recently awarded at the Ninth Asia-Pacific Computer Systems Architecture Conference (ACSAC'2004).