Lightweight Key Exchange and Stream Cipher based solely on Tree Parity Machines

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. Alternative security solutions are considered in science and industry, motivated by the strong restrictions as they are often present in embedded security scenarios – especially in a RFID setting. We investigate a low hardware-complexity cryptosystem for lightweight symmetric key exchange and stream cipher based on Tree Parity Machines. The speed of a key exchange is basically only limited by the channel capacity as is the stream cipher throughput. This work significantly improves and extends previously published results on TPMRAs. Again, characteristics of standard-cell ASIC design realizations as IP-core in 0.18μ-CMOS technology are evaluated.

Keywords: Embedded Security, Lightweight Symmetric Key Exchange, Lightweight Stream Cipher, Tree Parity Machine

1 Key Exchange, Stream Ciphers and RFID

The investigation of alternative security primitives and technologies is stimulated by the strong restrictions present in resource-limited devices. 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 if not available at all [1, 2, 3]. The RFID-industry should have a particular interest in security, because the commercial prosperity of their products is directly linked to the secrecy of data via customer acceptance [2, 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 in general [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), the generalization to Hyper-Elliptic Curves (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. Also, the ring-based asymmetric cryptosystem NTRU [8, 9] 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. As ECC requires more than 10000 gates and DES alone already demands a few 1000 gates, only lightweight stream ciphers are considered applicable for the extreme case of an RFID-tag with around 1000 gates and no microprocessor available (cf. [5]). Symmetric algorithms for this class are sought and stream ciphers are again considered for such niche applications [10]. Stream ciphers are regarded competitive with block ciphers when a small footprint in hardware implementations is required. Though the security aspects of RFID have not been standardized so far, the use of stream ciphers here seems forseeable due to the present constraints. Next to higher bandwidth, second generation RFID tags are planned to have improved security (encryption, password functions, authentication) and read/write capability. Key exchange requires read/write RFID devices for bidirectional communication and first tags with challenge and response authentication are developed [11, 12].

After all, a key exchange still remains of prohibitive cost and only stream ciphers seem to be applicable for encryption in strongly restricted domains. Seeking and investigating alternative approaches beyond efficient implementations of established primitives 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 and stream cipher based on so-called *Tree Parity Machines* [13]. We present a fully serial architecture-variant based on [14] using this key exchange concept and a trajectory mode, that allow for fast successive key generation and exchange, as well as for a synchronous stream chipher. It enables short key lifetimes through the achievable speed of a key exchange and consequently fast resynchronisation for the stream cipher. We focus on a low hardware-complexity IP-Core solution for resource-limited devices. 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 [13] as a method for symmetric key exchange. It does not involve large numbers and principles from number theory and is related to secret key agreement based on interaction over a public insecure channel as it is discussed under information theoretic aspects by Maurer and others [15, 16, 17, 18]. 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 unit receives different N inputs $(1 \leq j \leq N)$,

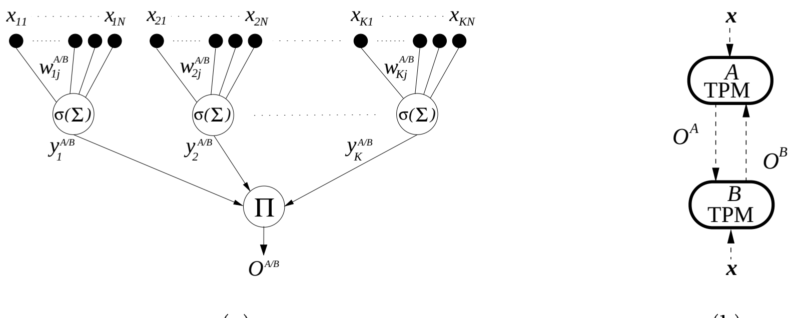


Fig. 1. (a) The Tree Parity Machine. A single output is calculated from the parity of the outputs of the summation units. (b) Outputs on commonly given inputs are exchanged between parties A and B for adaptation of their preliminary key.

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:

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

$\sigma(\cdot)$ denotes the sign-function. The so-called *bit package* variant (cf. [13]) 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 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 Equation 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 by privacy amplification [15, 16] 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.

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 (Equation 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 stated security weakness in [19], which assumes an ongoing communication. As soon as a new key is present, it is used for encryption. Each integer component of the the key is again XORred with an appropriate length concatenation of L input bits to further decorrelate subsequent keys. 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.

For the key exchange protocol *without* entity authentication, eavesdropping attacks have concurrently been proposed by Shamir et al. [20] and Kanter, Kinzel et al. [21, 22, 23]. But the prevalent definition of a successful attack is having a 98 percent average overlap $|w^E(t) \cdot w^{A/B}(t)|$ (averaged over all summation units) 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 authors chose this definition, because of the strong fluctuations in the success probability using a strict definition. The attacks in [20, 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 here does not know either, 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 [17, 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.

2.2 Key Exchange between Multiple Parties

Multiple parties can exchange a common key again based on TPM interaction and the synchronisation property. 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 TPM $p_{1,2}$. The exchange of a common key between $G > 2$ parties can thus 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.

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

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

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 practice only two TPMs in each group have to actively send and receive output bits, whereas the others in the group only receive.

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. This way, a linear chain

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

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 independantly be attacked, the security in the presented multi-party scenario scales inversly proportional to the number of parties.

3 Stream Cipher by Tree Parity Machines

A TPM stream cipher can be constructed as follows. Remember that once two parties are synchronous and successfully exchanged a key, they remain synchronous in each further iteration (trajectory mode) and produce equal outputs. The synchronous TPM stream cipher is based on the iteration of K coupled non-linear dynamic functions $y_k(t)$. The keystream generator can so be viewed as being composed of K dynamic filter generators and a final (static) combiner stage that acts similar to a threshold generator (see Figure 2). The initial state of the keystream generator depends on the key $w_{kj}^{A/B}(t_0)$ and the initialization vector $x_{kj}(t_0)$. Each dynamic filter generator consists of an N-bit LFSR (counter variables $x_{kj}(t)$) and of N L-bit up/down counters (U/D-CTRs) with a non-linear dynamic filtering stage. The filter depends on the key or current state (state variables $w_{kj}(t)$). The pseudo-random states of the LFSRs are expanded and mixed with the key state, pseudo-randomly modifiying signs of the key state. The subsequent integer addition (Equation 1) and reduction σ to a single sign (bit) $y_k(t)$ extracts the output of the dynamic filter generator. The final keystream output is an Exclusive-Or of K sign bits σ: $O(t) = \sigma_1 \oplus \sigma_2 \oplus \cdots \oplus \sigma_K$. The Exclusive-Or is also used as the statically balanced combiner to generate the ciphertext $c(t)$ from the keystream and the plaintext, i.e. $c(t) = O(t) \oplus p(t)$. The number of cycles to calculate one output bit (with the serial TPMRA) is $t_O = (K \cdot N + K) + 3$.

The next-state function

$$\phi_t : \mathbb{B} \times \mathbb{L} \times \mathbb{B} \times \mathbb{B} \mapsto \mathbb{L}, \quad \mathbb{B} = \{-1, 1\},\ \mathbb{L} = [-L, L] \subseteq \mathbb{Z} \tag{6}$$

$$\phi_t(O, w_{kj}, x_{kj}, y_k) \mapsto w'_{kj} \ , \tag{7}$$

defined via Equation 2, adapts and bounds the filter coefficients (the state) and represents a nonlinear state update, i.e. the keystream depends on a non-linear state-machine. As explained in Section 2, the state variables $w_{kj}(t)$ perform a random walk with reflecting boundaries in a state space of size $(2L+1)^{KN}$. The TPM stream cipher has $(2^{KN} - 1) \cdot (2L+1)^{KN}$ possible internal states divided into $K \cdot N$ state variables $w_{kj}(t) \in [-L, L]$ and the same number of counter variables $x_{kj}(t) \in \{-1, 1\}$.

Unlike other stream ciphers in output feedback mode (OFB), the keystream $O(t)$ is fed back to the next-state function and not to the LFSR (see Figure 2). As an alternative, the ciphertext bits can be fed back (CFB) instead of the

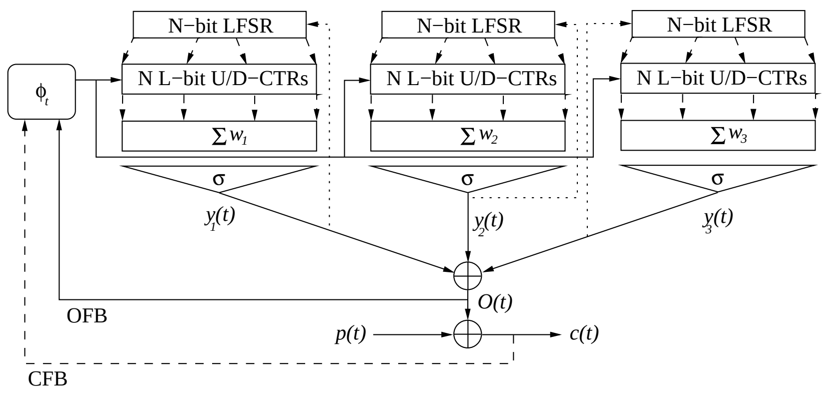


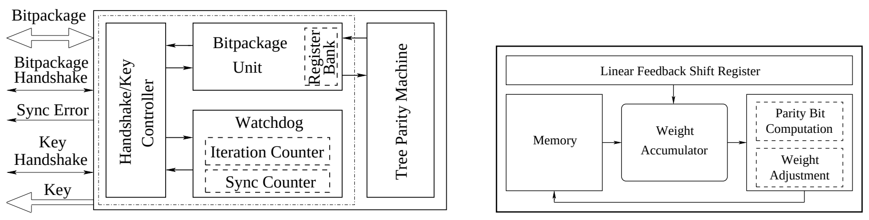
Fig. 2. The synchronous TPM stream cipher for $K = 3$. OFB and CFB modes are indicated. Alternatively, the LFSRs can also have an internal feedback from the summation and thresholding units (dotted lines) resulting in the socalled *Confused Tree Parity Machine* [25] with a simple shift register (with non-linear feedback) per hidden unit replacing the LFSR.

output bits (see Figure 2), making the internal state change also depend on the plaintext and yielding $\phi_t(c, w_{kj}, x_{kj}, y_k)$ in Expression 7. An integrity mechansim is present in the TPM stream cipher in CFB mode. Due to the feedback to the next-state function, a manipulation of the ciphertext leads to a change of the state update at the receiving side. A manipulated keystream thus leads to a decryption failure.

Often, in a real application of a stream cipher, it is required to use a single key many times but with a different initialization vector (IV). Using public initial values of the LFSRs, $2^{KN} - 1$ IVs can be chosen. Yet, the TPMRA allows for fast resynchronsation as a new key can efficiently be exchanged. Preliminary statistical analysis yield the keystream to be indistinguishable from random. Attacks on the stream cipher still have to be investigated.

4 ASIC-Implementation and Results

The *Tree Parity Machine Rekeying Architectures* (TPMRAs) [14] 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 3a gives an overview of the hardware structure. The Handshake/Key Controller Unit handles the key transmission (eventually 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



(a) (b)

Fig. 3. (a) Basic diagram of the Handshake/Key Controller with the Watchdog and the Bitpackage Unit. (b) The serial Tree Parity Machine Unit. The TPM controller state machine is omitted for clarity.

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 [13]. It reduces communication down to a few packages. In both architectures, the Bitpackage Unit partitions the parity bits (Equation 1) from the TPM Unit in tighter bit slices. In addition, it serializes the incoming bit packages from other TPM for the adaptation (Equation 2). The Bitpackage Unit handles bit package lengths up to n bits (depending on the key length).

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.

4.1 Serial TPM Unit

Different from the realization in [14], the serially realized TPM Unit calculates a parity bit serially in time 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 3b). 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 (Equation 1) and the Weight Adjustment Unit accomplishes the adaptation (Equation 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 register bank, 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 $t_{BP} = (2n-1) \cdot (K \cdot N + K) + 3$.

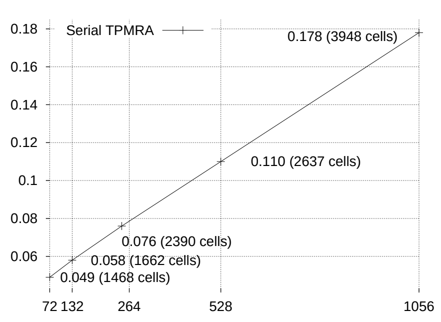
4.2 Results

Parameterizable serial TPMRAs were designed and simulated by using VHDL (compare [14]). While a FPGA-realization was used for easy prototyping, standard cell ASIC-realization prototypes were build to verify the suitability as an embedded system component. The underlying process is a 0.18μ six-layer CMOS process with $1.8V$ supply voltage based on the UMC library [26]. The linear complexity of the key exchange 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.

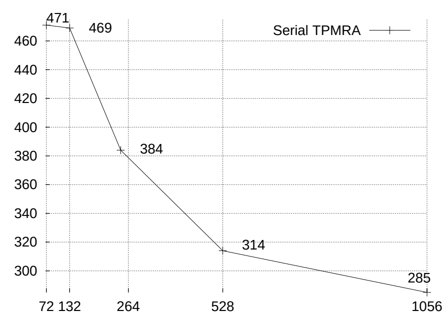
The cell-area (Figure 4a) of the serial TPMRA scales approximately linear due to the linear increase in required memory and ranges around 0.11 square-millimeter for the investigated key sizes. The number in braces denotes used standard cells. Note, that most of the area is consumed by the memory, because of the necessary storage of the partial results. The achievable clock-frequency (Figure 4b) ranges between 285 and 471 *MHz* for the investigated key lengths.

Additionally, we established the throughput for key exchange (i.e. keys per second) subject to the average synchronization time of 400 iterations for different key lengths in Figure 4c. 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 (interaction) overhead. This mode is identical to the stream cipher mode and we also appoint the theoretical throughput (bit-rate) of the TPM stream cipher which is in the *MHz*-range.

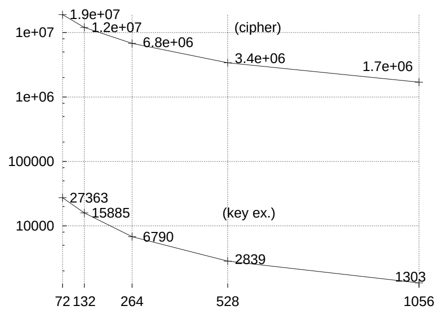
Figure 4d shows throughput regarding key exchange and stream cipher subject to a NFC and a RFID communication channel and their bandwidths. In key



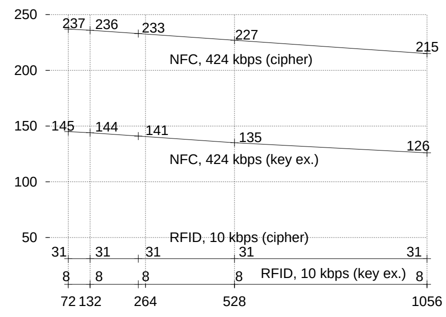
(a) Area [mm^2] vs. key length [bit]



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



(c) Average key exchange rate and stream cipher bit-rate [Hz] (log-scaled) vs. key length [bit] (idealized infinite channel bandwidth)



(d) Average key exchange rate and stream cipher bit-rate [Hz] vs. key length [bit] (NFC and RFID channel/protocol)

Fig. 4. Serial TPMRA post-synthesis area-optimized results (key exchange and stream cipher) vs. key length (UMC 0.18μ six-layer CMOS standard cell process).

exchange mode, for every protocol the minimum available packet length was used due to the necessary interaction through our bit packages of 32 bit: NFC (ECMA Intl. NFC IP-1) 136 bit and RIFD (TI TagIt-Protocol), 94 bit. For the RFID channel we appointed a 10 $kbps$ channel for simplicity. The capacities here vary with regard to Reader-to-Transponder (5-11 $kbps$) and Transponder-to-Reader (26 $kbps$) communication. In stream cipher mode, for every protocol the maximum available packet length can be exploited (NFC 359 byte with 255 byte payload, RFID 317 bit with 256 bit payload). A comparison among the different communication channels indicates different slopes of the calculated throughput characteristics (Figure 4d). They denote the rising influence of the output bit (bit packaging) calculations at smaller key lengths for channels of higher bandwidth. Thus, the slope of the NFC throughput characteristic is slightly higher than for RFID. As expected, the influence of the channel bandwidth significantly determines the performance of the key exchange protocol and of the stream cipher. Obviously, the bottleneck is the underlying communication-bus.

5 Conclusions

We suggest to discuss Tree Parity Machine Rekeying Architectures (TPMRAs) for low hardware-complexity lightweight authenticated symmetric key exchange and stream cipher. Efficient frequent rekeying (equivalent to a resynchronisation of the stream cipher) and short key lifetimes can be implemented. Next to using sophisticated encryption algorithms like Rijndal (AES), for example, performed with the exchanged key, the TPMRA itself allows for a lightweight stream cipher with feasible frequent rekeying. The stream cipher allows for a high throughput and is basically limited by the communication channel.

We regard the TPMRAs as IP-cores in embedded system environments with a particular focus on transponder-based applications such as RFID-systems, but also on devices in ad-hoc and sensor networks, in which a small area for cryptographic components is often mandatory. They are especially suited for devices of limited resources with no or only very limited microcontrollers available – even more in moderate security scenarios.

Acknowledgments

The authors would like to thank Sebastian Staiger (Hamburg University of Technology) for his contribution to the project, as well as University of Hamburg for allowing us to use their industry-standard synthesis-toolchain the ASIC design.

References

[1] 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
[2] Stanford, V.: Pervasive computing goes the last hundred feet with RFID systems. Pervasive Computing, IEEE Computer Science (2003) 9–14
[3] Sarma, S.E., Weis, S.A., Engels, D.W.: RFID systems and security and privacy implications. In Kaliski, B., ed.: Proc. of the Workshop on Cryptographic Hardware and Embedded Systems 2002, CHES 2002. Volume 2523 of LNCS., Springer-Verlag (2003) 454–469
[4] Weis, S.A., Sarma, S.E., Rivest, R.L., Engels, D.W.: Security and privacy aspects of low-cost radio frequency identification systems. In Hutter, D., ed.: Proc. of the 1st International Conference on Security in Pervasive Computing, SPC 2003. Volume 2802 of LNCS., Springer-Verlag (2004) 201–212
[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] Shamir, A.: Stream ciphers: Dead or alive? In: Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security – ASIACRYPT 2004. Volume 3329 of LNCS. (2004) 78 (see also Proceedings of SASC – The State of the Art of Stream Ciphers, Brugge, Belgium, 2004).

[11] Atmel Corporation: e5561 Read/Write transponder IC for contactless RF identification for highly sophisticated security applications. (2003) Datasheeet.

[12] Atmel Corporation: TK5561A-PP Read/Write Crypto Transponder for Short Cycle Time. (2005) Datasheet revision B.

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

[14] Volkmer, M., Wallner, S.: Tree parity machine rekeying architectures. IEEE Transactions on Computers **54** (2005) 421–427

[15] 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

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

[17] 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

[18] 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

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

[20] 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

[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] Ruttor, A., Kinzel, W., Shacham, L., Kanter, I.: Neural cryptography with feedback. Physical Review E **69** (2004) 7

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