CPU-side comparison for Key Agreement between Tree Parity Machines and standard Cryptographic Primitives

Domenico Lofù*†, Pietro Di Gennaro‡, Paolo Sorino*, Tommaso Di Noia* and, Eugenio Di Sciascio*

* *Dept. of Electrical and Information Engineering (DEI), Politecnico di Bari, Bari (Italy),*
e-mail: {domenico.lofu, paolo.sorino, tommaso.dinoia, eugenio.disciascio}@poliba.it

† *Innovation Lab, Exprivia S.p.A., Molfetta (Italy),*
e-mail: domenico.lofu@exprivia.com

‡ *Solutions and Support Unit (SSU), Client Solutions Delivery Section (CSDS), Service for Geospatial, Information and Telecommunications Technologies (SGITT), United Nations Global Service Centre (UNGSC), United Nations Department of Operational Support*
e-mail: pietrodig@live.com

***Abstract*—Information Security has become a crucial aspect nowadays in every domains. In order to protect these several domains, various cryptographic primitives have been implemented. To address this issue, in this paper we provide a key contribution. We compare three cryptographic primitives: Tree Parity Machines (TPM), Diffie-Hellman (DH) and Elliptic-curve Diffie–Hellman (ECDH) and show that TPMs is the best choice, based on cpu-side instructions, to make the key agreement between two counterparts A and B. Regarding DH and ECDH, tests have been performed using authenticated and unauthenticated versions. DH-Unified and Elliptic Curve Fully Hashed Menezes-Qu-Vanstone (ECFHQV) represent the authenticated version of DH and ECDH, respectively. We performed the comparisons for key agreement using programs compiled with a native programming language and a terminal tool to gather statistical information, such as some CPU-side evens, time required for synchronization, and information to be sent to the unsafe channel. Our detailed analysis, both formal and experimental, shows that cpu instruction-side TPM networks are ideal candidates for execute a key agreement between two counterparts A and B.**

***Index Terms*—Information Security, Neural Cryptography, Tree Parity Machines, Post Quantum Cryptography.**

I. Introduction

In the modern era, technological advancement is the main pillar. For instance, Artificial Intelligence (AI) is revolutionizing data management. Several fields, from medical to Internet of Things (IoT) [1]–[5], are constantly applying AI to solve various various daily tasks. In order to ensure the data remains secure and cannot be accessed by third parties, the transmission channel must be secured as well.

The science of cryptography is essential for the transmission and storage of data, as it ensures confidentiality, integrity, authenticity, and non-repudiation. Using cryptography, third parties cannot read or modify data in transit or on storage devices.

One of the basic steps in establishing a secure channel is the establishment of a secure key agreement. During key exchange, algorithms must ensure that no eavesdroppers can reproduce the secure key. The protocols for applied key agreements are based on mathematical operations that cannot be inverted computationally efficiently, for example factorizing large number problems.

Nowadays Integer Factorization, Discrete Logarithm Problems, and Elliptic Curves are three main mathematical problems.

In discrete logarithm problems, cryptographic structures are constructed and risks are exposed with quantum computing [6]. A variety of quantum-proof algorithms are available using modern cryptography techniques such as quantum cryptography [7] and neural cryptography[8], [9].

With Elliptic Curve Cryptography (ECC), smaller, faster, and more constrained public key systems can be implemented. Due to these features, ECC is positioned to meet low bandwidth and memory criteria. Conditional security is offered by current algorithms that are based on public-key cryptography.

The TPM [10] are considered. These are defined as two artificial neural networks that learn from each other until they reach a key agreement operations. This paper introduced a comparison in terms of CPU-side instructions, of the TPM, DH and ECDH family of cryptographic primitives in authenticated and unauthenticated versions.

Contribution. To this end, the first contribution of this paper is evaluation of the effort required, in terms of CPU-side instruction, to be able to make a key agreement between two counterparts, using TPM, DH and ECDH primitives. We compared authenticated and unauthenticated versions of the DH and ECDH algorithms families. We use *Prof Tool*[1] and we also consider a range of CPU-side instructions: CPU_Cycles, $CPU\_Instructions$, $Execution\_Time$ and $Bytes\_Sent$. Our results indicate that, the best approach to perform Key

[1] https://oopsmonk.github.io/posts/2022-04-28-perf/

Agreement between two counterparts, that predicts better values than the previous primitives, is with the use of TPMs.

Although cryptography is a research domain that has explored for many years, there is no work in the literature that performed a comparison like ours, based on cpu-side instructions. We believe that this work can make an important contribution to the entire scientific community. **Roadmap.** The remainder of the paper is organized as follows. Section II introduces the technical background related to Tree Parity Machines.
Section III illustrates the approach used in the experiments. In particular, Section III-A presents the experimental settings of our in-depth study on the comparison of cryptographic primitives, while Section III-B highlights some important evaluations based on the use of the *prof tool*, and Section IV concludes our paper.

II. Technical Background

In this section, we introduce briefly the DH and ECDH cryptographic primitives, that are employed to perform key agreement procedures. We also details the TPM architecture, used for realise the key agreement between two counterparts. We highlights the TPM Synchronization Procedure, Synchronization check and detail its operation. In addition, we cover a brief analysis of security and synchronization of this particular artificial neural network.

A. Diffie-Hellman Cryptography

DH key exchange, created by Diffie *et al.* [11], is considered one of the first public key exchanges implemented in the field of cryptography. It is based on the current difficulty of resolving the discrete logarithm problem. It allows two counterparts to establish a shared, secret key using an insecure (public) communication channel without the need for the two parties to have exchanged information. The key obtained through this protocol can later be used to encrypt subsequent communications using a symmetric encryption scheme.

B. Diffie-Hellman Elliptic Curve Cryptography

ECDH key exchange, created by Pelzl *et al.* [12], is anonymous key agreement scheme, more recent compared to DH, again based on the factorization problem of big primes. It allows two parties, each having an elliptic-curve public–private key pair, to establish a shared secret over an insecure channel. ECDH is very similar to the classical DH, but it uses Elliptic Curve Cryptography (ECC) point multiplication instead of modular exponentiations. ECDH is based on the following property of EC points: $(A * G) * B = (B * G) * A$. We considered two secret numbers A and B (two private keys, belonging to A and B counterparts) and an ECC elliptic curve with *generator point* G, we can exchange over an insecure channel the values $(A*G)$ and $(B*G)$ (the public keys of A and B counterparts) and then we can derive a shared secret: $secret = (A*G)*B = (B*G)*A$. Pretty simple. The above equation takes the following:

$$PubK_A * PrivK_B = PubK_B * Priv_A = secret \quad (1)$$

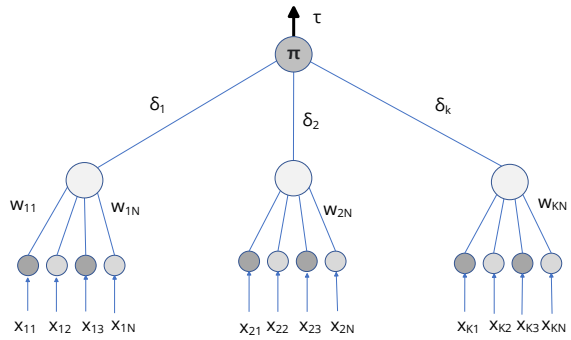

Fig. 1: Representation of the TPM structure.

C. Tree Parity Machines Architecture

The tree parity machine is a two-layered perceptron-structured artificial neural network with discrete weights, binary inputs, and binary outputs [13]. It consists of:

- $N * K$ input neurons;
- K hidden units;
- a single output neuron τ;

where N and K are integer values.
Fig. 1 represents the structure of the TPM with each element below described.
Inputs are integers:

$$x_{i,j} \in \{-1, 0, 1\} \quad (2)$$

While *Weights* between input and hidden neurons can assume the following values:

$$w_{i,j} \in \{-L, -L+1, \ldots, 0, \ldots, L-1, L\} \quad (3)$$

where the value L of var. (3) is defined *Synaptic Depth* and is an integer value.

Neuron's outputs are described by Eq. 4 and 5. Inputs, products, and weights are calculated as well as the sum of inputs and weights using their respective sign functions:

$$\delta_k = sgn(w_k \cdot x_k) \quad (4)$$

$$\delta_k = sgn(\sum_{j=1}^{N} w_{i,j} x_{i,j}) \quad (5)$$

Where the function sgn is defined as follows:

$$sgn(x) = \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases} \quad (6)$$

Finally, Eq. 7 denotes the *network's final value*, calculated multiplying the single values of all hidden units:

$$\tau = \prod_{i=1}^{K} \sigma_i \quad (7)$$

Where σ_i is defined as the $i-th$ hidden neuron.

1) TPM Synchronization Procedure: We assume that A and B are two counterparts (users, devices or counterparts). In addition, we assume that A and B both have the same TPM configuration and are trying to synchronize. The same TPM configuration exists when two counterparts have the same K, N, and L values, as well as the same learning rule to update the internal weights.

To synchronize correctly, they must expose their neural networks to the same common inputs (private or public) and exchange related outputs. The synchronization procedure is carried out through the steps in Algorithm 1:

Algorithm 1 TPMs Synchronization Procedure

procedure TPMSYNCH(A, B)
  $W_A(0) \leftarrow random$
  $W_B(0) \leftarrow random$
  while $W_A \neq W_B$ **do** ▷ If synch the procedure ends
    $t \leftarrow t + 1$
    $X \leftarrow random$
    $\tau_A \leftarrow output_A(X)$
    $\tau_B \leftarrow output_B(X)$
    if $\tau_A == \tau_B$ **then**
      $W_A(t) \leftarrow update\_rule(W_A(t-1), X)$
      $W_B(t) \leftarrow update\_rule(W_B(t-1), X)$
    end if
  end while
  $W_A = W_B$ ▷ Weights can be used as symmetric keys
end procedure

After the full synchronization is achieved (the weights $w_{i,j}$ for $i \in \{1, 2, \dots K\}$ and for $j \in \{1, 2, \dots N\}$ of both TPM are same), the counterparts can use their weights as keys. Regarding the synchronization, one of the following learning rules can be used:

- Hebbian learning rule:

$$w_i^+ = g(w_i + \sigma_i x_i \Theta(\sigma_i \tau)\Theta(\tau^A \tau^B)) \tag{8}$$

- Anti-Hebbian learning rule:

$$w_i^+ = g(w_i - \sigma_i x_i \Theta(\sigma_i \tau)\Theta(\tau^A \tau^B)) \tag{9}$$

- Random walk:

$$w_i^+ = g(w_i + x_i \Theta(\sigma_i \tau)\Theta(\tau^A \tau^B)) \tag{10}$$

Where: $\Theta(a, b) = 0$ if $a \neq b$ otherwise $\Theta(a, b) = 1$ and $g(x)$ is a function that keeps the w_i in the range $\{-L, -L + 1, ..., 0, ..., L - 1, L\}$.

2) TPM Synchronization check: An important aspect of the synchronization of TPM is understand how two machines are synchronized. This issue has not been widely explored in literature, except from Javurek *et al.* [14]. Knowing the exact point in which two TPM have the same weights is crucial, to avoid useless update operations and to not waste network resources, especially in IoT scenarios in which they should be saved. Trivially, the output of an hash function such as SHA256 could be used compare in a secure way the weights of the two machines, but this would not be optimized despite being secure and strong.

For this scenario, we considered an empirical value called AVG_SYNCH_STEPS, which uses a large number of tests as a reference (for example, 1000) and saves the minimal number of steps. During the main procedure, after *AVG_SYNCH_STEPS* steps in addition to the output of its respective machine, the result of the calculation of the following polynomial function, as described by Javurek *et al.* [14], is also sent:

$$P = \sum_{i=1}^{K} (\sum_{j=1}^{N} w_{ij}^{2k_j+1})^{2k_i+1} \tag{11}$$

Where P is the output of the polynomial function, w_{ij} are the value of the weights and k_i and k_j are exponents. In particular, the function is chosen to be secure against brute-force attacks. If the values of the polynomial sent corresponds for both the synchronizing TPM then the procedure ends.

3) TPM Synchronization and Security Analysis: Kinzel *et al.* [15] shown that simple attackers are always disadvantaged over the cooperating machines of A and B counterparts, because they do not influence the process of synchronization at all and cannot skip directly repulsive steps. In particular, we defined τ_A, τ_B and τ_E as output by the TPM A and B and by an eavesdropper Eve, respectively. The attacker will perform a weights update if and only if $\tau_A = \tau_B = \tau_E$.

If A and B disagree on the final output, it means that there is at least one hidden unit with $\sigma_t^A \neq \sigma_t^B$. An update of A and B would have repulsive effect and they just decide to do not update their weights. Accordingly, A and B have a clear advantage over E, because it cannot influence the synchronization process and she will be able to do it only slower than the active partners. Therefore bidirectional synchronization is faster than unidirectional learning [15], even if a small probability P_e still exists that the attacker will synchronize before the other twos and generally in real use case scenarios, one set this probability as an acceptable security level.

Common inputs are a key element to consider when designing systems that involve TPM and their synchronization [16]. In this regard, for example, public common inputs can expose the synchronization procedure to a family of possible attacks (such as the Geometric attack, Majority attack [17] or Genetic attack [18]) to intercept the internal weights that represent the information to secure between the synchronizing counterparts. If a secure authentication procedure is not implemented, Man-In-The-Middle attack represented a serious threat [16]. This security issue can be solved using implicit authentication by hiding common inputs, also with a Zero Knowledge authentication procedure [16]. However, an additional problem is presented: how to store, protect and distribute common secret input. Ruttor *et al.* [19] shown that with K hidden neurons, N inputs for every hidden units and the synaptic depth L, we obtain $(2L + 1)^{KN}$ combinations. For example, with $K = 3$, $L = 4$ and $N = 100$ we get ca. 10^{300} possibilities, which should be secure with the current silicon technology.

III. Proposed Approach

This section discusses a proposed comparison scheme. Specifically, Section III-A illustrates the experimental settings of our proposal, while, Section III-B provides performance evaluations to compare the CPU-side effort required to perform a key agreement operation with classical cryptographic primitives such as the DH algorithm or ECDH and TPM. Regarding on the use of TPM, we considered two counterparts A and B, that wish to establish a common secret through an communication channel which we assume to be public, insecure and shared.

A. Experimental Settings

Our experiments are based on*perf tool*, also known as Performance Counters for Linux (PCL). In particular, it is a profiler tool for Linux 2.6+ systems that abstracts away CPU hardware differences in Linux performance measurements and offers an intuitive command-line interface. In addition to executing native applications, this tool also gathers performance counter statistics, such as cycles and instructions retired.

The *perf tool* software only supports compiled programs, specifically, applications written in C or C++, which are natively compiled. Since TPM are implemented in pure C code, the comparison of the key agreement procedures between DH and ECDH families uses C++ code using *Crypto*++ library [20].

Despite it is possible to write pure C/C++ vanilla implementations in native code, the suggested configurations of the algorithms used to perform a key agreement often require extra efforts, which generally means using a prime group of at least 3072 bits for the DH family, or an elliptic curve with a minimum size of 256 bits for the ECDH one, all of which are transported using an AES key of at least 128 bits. According to this, in our experiments values from *RFC-5114* [21] have been used for the DH key agreement software, while for the other family of ECDH configurations have been randomly initialized, though using a 256 bits curve.

To compare the effort required to reach a key agreement between primitives, the *perf tool* instructions are as follows:

- *CPU_Cycles:* The number of CPU cycles associated with an executed program. Cycle time refers to the time required to execute one simple processor operation, such as an addition.
- *CPU_Instructions:* Count of CPU instructions required to process the program.
- *Execution_Time:* The target program execution time, measured in seconds.
- *Bytes_Send:* The amount of information counterparts A and B must exchange in order to conclude their key agreement. DH and ECDH families use public keys exchanged over unsecure channels. In the case of TPMs, this is the total number of bits produced by both synchronizing machines; for TPMs, it is the number of bits produced by both synchronizing machines.

In our experiments TPMs are configured with the following parameter: $K = 3$, $N = 50$ and $L = 4$. In this scenario, the only attack possible would be a brute-force attack, where an attacker has a probability of $(2L + 1)^{KN}$, which for the configuration considered would be ca. 10^{143}. We have hidden the common inputs of the TPM in order to eliminate all proposed attacks against them and to provide authentication, since an attacker can now only access the information regarding the machines' outputs.

B. Evaluation

In Table I the summary of all the experiments are listed. The comparisons include the authenticated and unauthenticated versions of the two considered cryptographic families, DH and ECDH, where the formers are obviously stronger against Man in the Middle (MITM) attacks, but also requiring more information to be exchanged over the unsecure channel. The considered authenticated algorithms are DH Unified (also known as Ephemeral Unified Model Scheme or Static Unified Model)for the DH [22] family and the Elliptic Curve Fully Hashed Menezes-Qu-Vanston (ECFHMQV) [23], [24] for the ECDH family.

Generally speaking, authentication is provided in several ways, for example using certificates, where every certificate is basically a wrapper around a public key, also containing digital signatures and other information connected with the key, or using Zero Knowledge (ZK) protocols. Authenticated protocols often require the publication of a static public key that acts in the long term in the same way certificates do, so they need to be retrieved once and this has been considered in the *Bytes_sent* column of Table I.

TPM related results for our experimental setup are average values of 100 different machines synchronizations. As a result of the stochastic initialization of both the weights and the common inputs of A and B counterparts in Neural Cryptography, the agreement process is not deterministic.
Based on the results provided, Table I indicates that TPM have a significant advantage over DH and ECDH primitives for both authenticated and unauthenticated key agreements in terms of operations performed, execution time, and network payloads. The TPMs results, report CPU_Cycles, $CPU\_Instructions$, $Execution\_Time$ and $Bytes\_Sent$ values of 5785115, 12196906, 0.00304 seconds and 28 bytes, respectively. The DH results, report $Cycles$, $Instructions$, $Execution\_Time$ and $Bytes\_Sent$ values of 42740964, 88273399, 0.01857 and 768, respectively, while ECDH report values of 20929224, 50155510, 0.0094 seconds and 128 bytes.

From the experimental results shown in Table I, it follows that results were largely predictable, since from the definition of TPM it's clear that only simple number theory operations are executed during the process of output calculation and weights update. The important downside of the TPM synchronization process is the continuous communication requirement, according to which the machines should exchange and compare their outputs when exposed to common inputs. This is important because TPM synchronization could be considered not efficient when information are exchanged using high level application protocols, such as TCP or UDP, where every single machine output bit should travel with the protocol payload.

Furthermore, the data in the table shows that the important data are instructions and CPU cycles of execution: in all cases,

Cryptographic_Primitive	Authenticated	CPU_Cycles	CPU_Instructions	Execution_Time	Bytes_Sent
TPM	✓	5785115	12196906	$0.00304s$	$28B$
DH	×	42740964	88273399	$0.01857s$	$768B$
DH Unified	✓	43696463	$91920943s$	0.0415	$1536B$
ECDH	×	20929224	50155510	$0.0094s$	$128B$
ECFHQV	✓	22258822	50825863	$0.0128s$	$256B$

TABLE I: Summary of the comparisons between unauthenticated and authenticated classic cryptographic primitives and TPM

it founded that TPMs offer performance and authentication advantages, performing fewer operations and guaranteeing authentication.

As a result of the simple operations required by TPM synchronisation, this advantage is very useful in systems with limited resources and architectures, such as IOT or Swarm Robotic.

IV. Conclusion and Further Directions

In this paper, we proposed a CPU-side comparison of the effort required to perform a key agreement between TPM and classical cryptographic primitives such as DH and ECDH. Comparison was performed using the *perf tool*. In particular, the CPU-side instructions used, were: *CPU Cycles*, *CPU Instructions*, *Execution Time* and, *Byte Send*.

According to our results, TPM is the best cryptographic primitive. During the experimental phase, it was discovered that TPMs were more efficient in terms of information sent and processed between counterparts, compared to other cryptographic primitives considered. The artificial neural network reports the following performance: $Cycles$, $Instructions$, $Execution\_Time$ and $Bytes\_Sent$ of 5785115, 12196906, 0.00304 seconds and 28 bytes respectively.

Finally, TPMs can be authenticated by hiding common inputs: neural networks exposed to different inputs do not sync.

During the experiments, it was not possible to separate the two synchronizing programs connected to A and B counterparts into separate processes. A zero network latency, zero protocol overhead, and zero network latency ideal insecure channel was considered for the comparisons. A future solution will be to use local sockets, pipes, or messages to exchange information between two separate processes.

Furthermore, we propose to use in the experimental phase, Industrial IoT Integrator Board (e.g. *Arduino*[2] or *Raspberry*[3]) to verify the feasibility and efficiency of the various implementations in contexts with extremely limited resources.

In this paper, there were authenticated and unauthenticated versions of the two cryptographic families DH and ECDH compared. In future work, RSA and public key cryptography algorithms can be included in the comparison.

Furthermore, to solve the inefficient synchronization problem (mentioned in Section III-B), we propose to develop the TPM *bit package* variant. In this variant, an array of $b > 1$ TPM outputs is sent, reducing the exchanged message by orders of magnitude.

We also released the source code of this comparison[4], to enable practitioners, industry, and academia, to use our solution as a ready-to-use basis for further investigations.

Acknowledgment

The authors would like to thank the Apulia Region for Financial Support. This work was partial support of the projects: Italian P.O. Puglia FESR 2014 – 2020 (project code 6ESURE5) 'SECURE SAFE APULIA', Fincons CdP3, PASSPARTOUT, Servizi Locali 2.0, ERP4.0 and, CTEMT - 'Casa delle Tecnologie Emergenti di Matera'.

References

[1] P. Boccadoro, V. Daniele, P. Di Gennaro, D. Lofù, and P. Tedeschi, "Water quality prediction on a sigfox-compliant iot device: The road ahead of waters," *Ad Hoc Networks*, vol. 126, p. 102749, 2022.

[2] E. Lella, A. Pazienza, D. Lofù, R. Anglani, and F. Vitulano, "An ensemble learning approach based on diffusion tensor imaging measures for alzheimer's disease classification," *Electronics*, vol. 10, no. 3, p. 249, 2021.

[3] C. Ardito, T. D. Noia, E. D. Sciascio, D. Lofù, A. Pazienza, and F. Vitulano, "User feedback to improve the performance of a cyberattack detection artificial intelligence system in the e-health domain," in *IFIP Conference on Human-Computer Interaction*. Springer, 2021, pp. 295–299.

[4] C. Ardito, I. Bortone, T. Colafiglio, T. Di Noia, E. Di Sciascio, D. Lofù, F. Narducci, R. Sardone, and P. Sorino, "Brain computer interface: Deep learning approach to predict human emotion recognition," in *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2022, pp. 2689–2694.

[5] S. Aresta, I. Bortone, F. Bottiglione, T. Di Noia, E. Di Sciascio, D. Lofù, M. Musci, F. Narducci, A. Pazienza, R. Sardone *et al.*, "Combining biomechanical features and machine learning approaches to identify fencers' levels for training support," *Applied Sciences*, vol. 12, no. 23, p. 12350, 2022.

[6] Z. Kirsch and M. Chow, "Quantum computing: The risk to existing encryption methods," *Retrieved from URL: http://www. cs. tufts. edu/comp/116/archive/fall2015/zkir sch. pdf*, 2015.

[7] C. H. Bennett, F. Bessette, G. Brassard, L. Salvail, and J. Smolin, "Experimental quantum cryptography," *Journal of cryptology*, vol. 5, no. 1, pp. 3–28, 1992.

[8] A. Klimov, A. Mityagin, and A. Shamir, "Analysis of neural cryptography," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2002, pp. 288–298.

[9] W. Kinzel and I. Kanter, "Neural cryptography," in *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP'02.*, vol. 3. IEEE, 2002, pp. 1351–1354.

[10] M. Volkmer and S. Wallner, "Tree parity machine rekeying architectures," *IEEE Transactions on Computers*, vol. 54, no. 4, pp. 421–427, 2005.

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

[12] J. Pelzl, T. J. Wollinger, and C. Paar, "Low cost security: Explicit formulae for genus-4 hyperelliptic curves," in *Selected Areas in Cryptography*, 2003.

[13] S. Marsland, *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC, 2011.

[2] https://docs.arduino.cc/retired/boards/arduino-industrial-101

[3] https://www.embeddedpi.com/integrator-board

[4] https://github.com/domenicolofu/CSC_KA

[14] M. Javurek and M. Turcanik, "Synchronization verification improvement of two tree parity machines using polynomial function," in *2018 New Trends in Signal Processing (NTSP)*, 2018, pp. 1–5.
[15] W. Kinzel and I. Kanter, "Interacting neural networks and cryptography," 04 2002.
[16] M. Volkmer and A. Schaumburg, "Authenticated tree parity machine key exchange," *IACR Cryptol. ePrint Arch.*, vol. 2004, p. 204, 2004.
[17] A. Ruttor, W. Kinzel, and I. Kanter, "I.: Neural cryptography with queries," *Journal of Statistical Mechanics Theory and Experiment*, vol. 2005, 12 2004.
[18] A. Ruttor, W. Kinzel, R. Naeh, and I. Kanter, "Genetic attack on neural cryptography," 12 2005.
[19] A. Ruttor, W. Kinzel, and I. Kanter, "Dynamics of neural cryptography," *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 75, p. 056104, 06 2007.
[20] "The Crypto++ library," https://www.cryptopp.com, 2022, (Accessed: 2022-11-22).
[21] M. Lepinski and S. Kent, "Additional Diffie-Hellman Groups for Use with IETF Standards," IETF, RFC 5114, Jan. 2008. [Online]. Available: http://tools.ietf.org/rfc/rfc5114.txt
[22] E. Barker, L. Chen, A. Roginsky, A. Vassilev, and R. Davis, "Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography," 2018-04-16 2018.
[23] A. P. Sarr, P. Elbaz–Vincent, and J. Bajard, "A secure and efficient authenticated diffie–hellman protocol," Cryptology ePrint Archive, Paper 2009/408, 2009, https://eprint.iacr.org/2009/408. [Online]. Available: https://eprint.iacr.org/2009/408
[24] H. Krawczyk, "Hmqv: A high-performance secure diffie-hellman protocol," Cryptology ePrint Archive, Paper 2005/176, 2005, https://eprint.iacr.org/2005/176. [Online]. Available: https://eprint.iacr.org/2005/176