Voting System Design Pitfalls: Vulnerability Analysis and Exploitation of a Model Platform

Kelvin Ly, Orlando Arias, Jacob Wurm, Khoa Hoang, Kaveh Shamsi, and Yier Jin
Department of Electrical and Computer Engineering, University of Central Florida
yier.jin@eecs.ucf.edu

***Abstract*—Homomorphic encryption may be seen as a substantial potential boon to voting systems. If properly used, it allows provably anonymous elections to take place. However, when poorly constructed, using weak cryptographic primitives results in highly vulnerable systems that are prone to attacks. This paper details one attack done against a model of an election system as part of a security competition, where a hardware Trojan has weakened its security. We designed a proof of concept exploit and implemented it on an FPGA, demonstrating weaknesses in the system regardless of the existence of this Trojan.**

I. Introduction

The emergence of hardware Trojans has largely reshaped the traditional view that the hardware layer can be blindly trusted [1]. Hardware Trojans, which are often in the form of maliciously inserted circuitry, may impact the original design by data leakage or causing circuit malfunctions. The CSAW 2015 Embedded Security Challenge involved exploiting vulnerabilities in a specified voting system [2]. The system, at a surface level, does follow good cryptographic protocol, but is marred and made vulnerable at implementation level.

For the purposes of this challenge, the organizers provided the election system. In order to ensure the confidentiality and integrity of the election results, the system incorporates the newly designed "AHE-1" cryptoprocessor. This cryptoprocessor implements authenticated homomorphic encryption based on the Paillier cryptosystem [3] and "AHE" fast message authentication code (MAC) over a sequence of concatenated ciphertexts. The system follows the Encrypt-then-MAC paradigm for authentication and integrity checking.

Though this system is said to be secure, we work under the assumption that as designers we added a hardware Trojan into the cryptographic processor which went undetected. The Trojan lies in implementation of the Paillier Key generation algorithm. Instead of generating cryptographically secure primes, it generates primes between 2^{16} and 2^{28}, greatly reducing the security of the system.

II. System Specifications

In this system, voting machines report their votes as unspecified intervals back to a master server, which verifies the message using its message authentication code (MAC), before applying the payload to the vote numbers. The vote counts are encrypted using the Paillier cryptosystem with a weakened key, providing privacy for its users. The use of a MAC appended to each message provides authentication by requiring the sender to know the associated secret, and also integrity by allowing the server to verify the contents of the message.

The Paillier cryptosystem is the most innovative part of the system. The Paillier cryptosystem is homomorphic, allowing the vote additions to be anonymous; the server can add the votes without decrypting the values, providing privacy for the users [3]. Homomorphic cryptosystems are cryptosystems where that the encrypted values can be operated on, generally in a limited way, to perform computations on their equivalent decrypted values [4]. The vote counts are encrypted using Paillier, and then are tallied together while still encrypted. The results are decrypted at the end of the election, where the winning candidate is declared. Previous work has demonstrated that this is a provable secure way to provide private voting [5]. Since the tallying server does not use the private key to perform addition of ciphertexts it can be made entirely oblivious to the value of individual vote counts. Additionally, the Paillier cryptosystem is a form of public key cryptography which allows secrecy of communications over an untrusted channel.

The messages are sent as Ethernet II DIX frames, using a destination media access control (MAC) address of `FF:FF:FF:FF:FF:FF`, broadcasting to all nodes in the network. Besides the frame type, these messages contain the public parameter of the encryption key (`SECPARAM`), as well as a series of two-tuples with the candidate identification and the encrypted vote count. The messages are padded and cryptographically signed using a Merkle-Damgård chain [6]. Furthermore, the messages must be within a certain length range; the server will reject the frame if the payload length is not between 46 and 1500 bytes long.

III. Vulnerabilities

Under normal circumstances, the message authentication code (MAC) attached to the message ensures the validity and integrity of the message. Also, the usage of a Merkle-Damgård chain ensures that as long as a collision resilient compression function is used, a collision resilient hash function can be built [6]. However, the proposed voting system utilizes a 16-bit key and a 16-bit initialization vector as seeds of the hash function. Exacerbating the problem is that the chosen block size for the compression function is also 16-bit in size.

Under the Merkle-Damgård construct, the key k is prepended to the message M. The message is divided into blocks m' with padding added to the last block if required. The message is then passed through a compression function $h(h', m')$, where h' is the chaining variable. Initially, the chaining variable is set to the initial vector, $h'_0 = iv$. The first computation of the compression function is then done to find the next element in the chain. This computation is done against the first block of the message. The first block contains the key, thus $h'_1 = h(iv, k)$. Subsequent iterations over the Merkle-Damgård chain yield the next element in the chain: $h'_{n+1} = h(h'_i, m'_i)$. The computation of the hash finishes when the last block m'_n has been passed through the compression function $h'_{n+1} = h(h'_n, m'_n)$, with h'_{n+1} being the resulting MAC of the message.

In the election system, both k and iv are preshared between the tally server and the voting machines. Since the attacker does not know signature parameters, these must be computed. This proves to be inexpensive: k and iv are the first inputs to the compression function $h(h', m')$, which results in the output of a 16-bit block that is carried over as the chaining variable h'. As such, an attacker only needs to find a 16-bit number to alter the message. Brute forcing the result of $h(iv, k)$ proves to be trivial even in general purpose computing systems. Once the initial block has been computed, the message can be altered at will. Furthermore, since the key and the initialization vector are preshared, these can not be changed during the election even when compromised.

The second vulnerability in the election system is a direct product of the hardware Trojan that was inserted during its design. The public parameter N is calculated as the product of two randomly generated prime numbers. However, the prime numbers generated by the "AHE-1" cryptoprocessor are greatly reduced in size, to a range between 2^{16} and 2^{28}. Although there is no known solution to the integer factorization problem in polynomial time, factorizing N under these constrains proves to be computationally feasible. Utilizing the prime counting function, we see that there are about $\pi(2^{28}) - \pi(2^{16}) \approx 13.83 \times 10^6$ primes in that interval. The factorization of N then becomes trivial, especially since this amount of numbers can easily be stored.

IV. High Level System Overview

Our system is implemented on the Nexys 4 DDR FPGA. Figure 1 provides a diagram of the working procedure of our implementation. Since Ethernet frames are broadcast in the network, our system captures then and parses their contents. We extract the `SECPARAM` which gives us a portion of the public key of the Paillier cryptosystem. We also extract the message authentication code (MAC) from the frame and proceed to brute force the MAC key and initialization vector parameters of the Merkle-Damgård chain. To avoid possible collisions, multiple frames are required to complete this process. We utilize extra frames to test the possible $h'(iv, k)$ combinations we computed, validating them against the extra frames.

Simultaneously, the system begins to factorize the `SECPARAM` field to compute the private key. Since the prime number generation in the cryptographic processor has been severely weakened, only numbers in the range of 2^{16} to 2^{28} need to be tested as factors. We iterate through all primes in this interval testing N for divisibility against each prime. Once

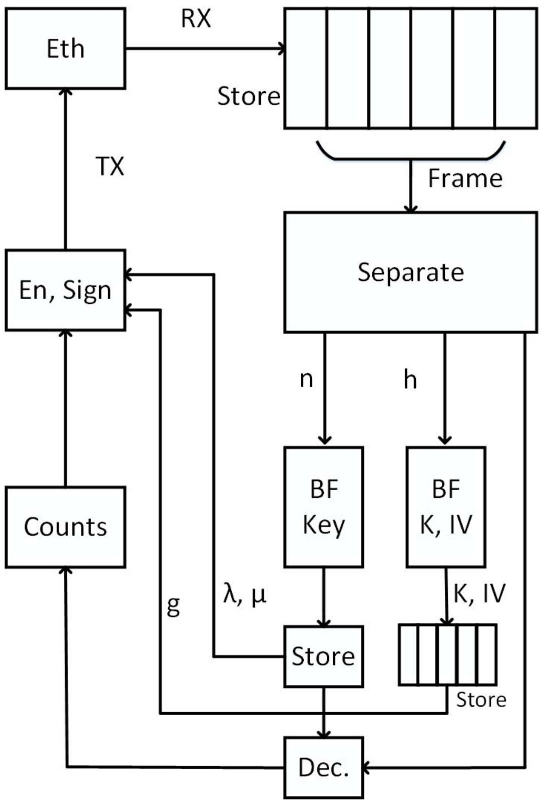

Fig. 1: High Level Overview of FPGA implementation

the prime factors are found, the system moves on to computing the private parameters of the private key, λ and μ.

Once the private key has been obtained, the system can track the number of votes each candidate has received. We obtain the number of votes and decrypt them from the two-tuples that are sent from the Ethernet frames. All received frames are analyzed and votes are tallied from them. We keep the votes correlated to the candidate identification numbers to measure which candidate is winning and its margin.

The system must also keep track of the amount of votes the tally server has received, as if the amount is too small or too high, the tally system will declare the election invalid. The system keeps track of how many votes each candidate has received and will modify the election votes while keeping the total amount of votes the same. This works by adding negative values of the votes for the other candidates, while adding the same amount of votes for our candidate. Done correctly, this should result in a perfect 100% election for our candidate.

As the system obtains the vote counts per candidate, it influences the result of the election by crafting its own set of Ethernet frames and broadcasting them on the network. Following the frame format specified by the voting system, a new frame is generated containing a malicious payload that increments the votes of our selected candidate while resetting the votes of the rest. This last action is performed as a way to not trigger suspicion based on the total number of votes received. The MAC for the frame is computed and appended to the message. Since the server is able to authenticate the frame and does not decrypt it as to ensure anonymity, it is unable to detect the attack. As a result, when the election finishes, our candidate wins with all the votes.

V. Implementation details

A. Initial Structure

Instead of using a soft CPU core to aid with some of the logic, the entire project was implemented using Verilog HDL modules. The majority were written with this project in mind, the exception being the SD card reader, which was obtained from XESS Corporation [7]. A hardware-only mechanism means that more components can be working in parallel, potentially improving the performance of the system.

The smaller computational modules, requiring high efficiency and speed, are synchronous with respect to communication. Given an input, the result would be expected at the output after a constant number of cycles, no other communication necessary. Modules in this category include the modular multiplication module, which uses the Kochanski modular multiplication algorithm [8], the hashing function, counters, and a few others. Karabutsa multiplication [9] was used in the prime number factorization and generation of private keys.

The larger modules, which are composed from these simpler modules, utilize a simple form of asynchronous communication. These larger modules are all developed as finite state machines, which can be optimized by the Xilinx toolchain, and are easy to reason about. All inputs and outputs have two additional wires, `vld` and `ack`. The transmitting side of the port asserts `vld` and keeps it high until the receiving side asserts `ack`, signaling a completed transaction. This simple protocol allows these modules to ensure only valid values are received by other units.

The modules which break the hash function parameters are somewhat complicated by the fact that it is generally unfeasible to determine the proper key k and initialization vector iv from a single frame. Due to collisions in the hash function, multiple $h(iv, k)$ can generate the same signature. As a result, two memory blocks were used to store a sorted list of all possible $h(iv, k)$. As a frame is received, a new block is used. The data in the two blocks are merged as an intersection of each other, narrowing the number of possible valid $h(iv, k)$ in record. The operation completes when the list holds a single element, which becomes the seed for all future message authentication code (MAC) computations.

Two FSMs are provided to write and receive data from the LAN8720A Ethernet PHY chip in the Nexys 4 DDR board. During the period where we were testing this portion of the core, Wireshark, a tool capable of capturing network packets down to the physical layer, proved integral in debugging this interface [10]. The Ethernet controller runs at 50 MHz, instead of the 100 MHz used across the rest of the FPGA design, so asynchronous buffers were used to interface between an FSM in the 50 MHz domain and one in the 100 MHz domain to allow for valid clock domain crossing. The Ethernet connection handler was given a buffer, so that the device would be able to continue receiving packets while it was still processing existing packets. The core effectively swaps between two Ethernet handlers, each with its own 2 KB buffer. This means that one additional packet could be received while the current one is being processed. For demonstration purposes this was considered sufficient, as it was expected that this would be a relatively low traffic system. As such, the resulting system can process new packets after the initial one can be processed as fast as they can be received.

A similar system is used to store the decrypted vote counts. Content-addressable memory was considered as the storage medium but the area cost was deemed too high for its benefits. As such, a simpler random access memory was selected where the candidate identification-vote count two-tuple is stored as an array. Although this meant searching for a candidate would take linear time, this operation contributes a negligible cost compared to the other calculations involved. Since other operations which occur in parallel take longer, the linear search time is masked by the other processes.

The Paillier cryptosystem involves modular exponentiation. This is implemented using the standard square and multiply algorithm, using the Kochanski modular multiplication module as the base primitive. The Paillier encryption and decryption modules consumed the most area of all the components, due to the need to add large numbers, which required extensive LUTs and routing resources.

B. Optimizations

The hash function required four clock cycles to calculate a hash from its two components. A naive implementation would consequently require approximately 4×32768 clock cycles to find the compressed key and initialization vector $h(iv, k)$. However, because the hash function was designed as a pipeline, it is possible to feed in four inputs consecutively. The brute forcing finite state machine iterates through the possible keys using a simple counter, so it is simple to determine the values to input into the function each cycle. This allows a throughput of one hash/clock cycle to be achieved, reducing the time by a factor of four. Further parallelism was possible, but the performance at this point was considered sufficient; 32768 cycles is a fraction of a second at 100 MHz.

The prime number factorization required all the prime numbers from 2^{16} to 2^{28}. The list of prime numbers in this range is stored on an SD card. The alignment of the data as well as its size has a direct impact the performance of the system. Naively storing the numbers would then require a 4-byte alignment and a 4-byte data transfer for each prime number. However, we improve the storage and transfer requirements by utilizing a simple form of delta compression. We first note that the difference between two primes in this range is necessarily an even number. We also notice that the difference is never greater than 512. As such, we store half the difference, thereby discarding an implicit least significant bit of 0 in the difference and reducing our storage requirement to 1 byte per prime number. This results in a 75% reduction in storage needs as well as in data transfers.

VI. Testing and Results

Testing, both in simulation using Icarus Verilog and on the FPGA, demonstrated that all the components separately worked correctly. However, integration of all the components was not finished in time for competition. Each individually tested component performed its task in a time efficient manner, with most of them clocking times of under a second. As such, the attack could override the election results in the extremely small timeframes.

We were able to successfully brute force the compressed key and initialization vector $h(iv, k)$ utilized in the Merkle-Damgård chain of the message authentication code (MAC) algorithm. Because of the size of the search space (only 2^{16} entries), this procedure takes less than a second, even in a general purpose computing system. Once we found the correct key and initialization vector, we were able to create our own signatures for our Ethernet frames. This type of attack is possible given the preshared key nature of the election system.

We were also able to successfully brute force the factors of the public parameters N of the Paillier scheme. Utilizing the precomputed table of primes in the range of 2^{16} to 2^{28}, we were able to compute the factors of N by performing division by the prime numbers in the table. This process took consideraby longer, totalling about 5 seconds, because of the overhead in accessing the SD card. Prime factorization of the public parameter N was crucial in computing the private key parameters, μ and λ. Once we computed the private key, we were able to keep a record of each candidate's votes by decrypting the incoming Ethernet frames.

A fully software-based version was developed simultaneously to ensure that the system could work in theory. The software-based implementation was fully integrated and was capable of overthrowing the results of the election with no user interaction. The main delay in the software implementation was in the factorization of the public parameter N, which is needed only once per election[1].

VII. Solutions for found vulnerabilities

The two key faults in the system were the poor choice of message authentication code (MAC) algorithm and the weak public/private key pair. The former issue is more important to fix, as integrity and authentication should be more important than the privacy afforded by the latter. Using the standard hash-based message authentication code (HMAC) [11] alongside a stronger compression function with a larger block size would drastically improve the security of the system. Cryptographically secure encryption functions can be used as hash functions under some circumstances, such as AES in Galois Counter Mode. In the same vein, reusing keys and initialization vectors on every message adds to the possibility of a replay attack. Keys and initialization vectors should be negotiated with the server prior to any transfer using standard protocols, such as the Diffie-Hellman-Merkle key exchange [12]. With these improvements in mind, a cryptographically weaker Paillier key has a lesser detrimental effect on the overall security of the system, although privacy can be lost to an attacker. A cryptographically secure key size would ameliorate the latter issue.

[1]Since this solution was outside the scope of the competition, it was not presented and was only used as means to check the sanity of the hardware implementation.

VIII. Conclusions

Overall, the election system proposed in this competition is flawed in many ways. One of the biggest takeaways is that embedded devices should be tested more thoroughly. There are many devices that are currently deployed on the market which were not properly tested and leak sensitive information. In addition, the implementation of the server in this challenge is not secure, as it accepts information from any node that can compute the correct HMAC. This implementation, though obviously insecure, is representative of many modern implementations of servers that communicate with embedded devices. Many servers accept incoming connections on specific ports without authenticating the connection because how uncommon the services provided in those ports are. Sometimes, these services running on the ports are not secure, and as such can leak information. As for future work, we will investigate the security of other mission-critical cyber-physical system applications in order to protect these systems.

Acknowledgments

This work was supported in part by the Florida Cybersecurity Center (FC2) 2015 Collaborative Seed Grant Program and the Southeastern Center for Electrical Engineering Education (SCEEE 15-001). Mr. Orlando Arias and Mr. Khoa Hoang are partly supported by the REU Supplement of an National Science Foundation grant (CNS-1319105).

References

[1] Y. Jin, "Introduction to hardware security," *Electronics*, vol. 4, no. 4, pp. 763–784, 2015.

[2] *https://github.com/nekt/csaw_esc_2015*.

[3] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 223–238.

[4] O. Mazonka, N. G. Tsoutsos, and M. Maniatakos, "Cryptoleq: A heterogeneous abstract machine for encrypted and unencrypted computation," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 9, pp. 2123–2138, 2016.

[5] E. Cusell Sánchez, "Homomorphic cryptosystems for electronic voting," 2014.

[6] R. C. Merkle, R. Charles *et al.*, "Secrecy, authentication, and public key systems," 1979.

[7] "XuLA FPGA Board Repository," 2015, https://github.com/xesscorp/XuLA.

[8] M. Kochanski, "Developing an rsa chip," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1985, pp. 350–357.

[9] A. KARABUTSA and Y. OFMAN, "Multiplication of many-digital numbers by automatic computers," *DOKLADY AKADEMII NAUK SSSR*, vol. 145, no. 2, p. 293, 1962.

[10] G. Combs *et al.*, "Wireshark," *Web page: http://www.wireshark.org/*, pp. 12–02, 2007.

[11] H. Krawczyk, R. Canetti, and M. Bellare, "Hmac: Keyed-hashing for message authentication," 1997.

[12] M. E. Hellman, B. W. Diffie, and R. C. Merkle, "Cryptographic apparatus and method," Apr. 29 1980, US Patent 4,200,770.