Ternary Computing to Stengthen Cybersecurity

Development of Ternary State based Public Key Exchange

Bertrand Cambou
Northern Arizona University
Bertrand.cambou@nau.edu

Donald Telesca
Air Force Research Laboratories
Donald.telesca@us.af.mil

Abstract— **General purpose ternary computers have not been successful so far, because they are not as efficient as the binary ones to compute binary codes. However, heterogeneous computing systems with reduced instruction set computing (RISC) engine dedicated to crypto-system, do have the potential to greatly strengthen cybersecurity, while keeping legacy binary codes on binary engines to maintain performance. In this paper, we are presenting the development of a Ternary based Public Key exchange scheme that is Addressable (T-PKA) to replace, or complement, the existing Public Key Infrastructures (PKI). In this scheme, the initialization step, also called "personalization", is based on the secure exchange of addressable cryptographic tables between the communicating parties. The content of these tables is generated either with ternary random numbers, or with arrays of addressable ternary PUFs. Private keys are generated independently with the shared public keys by all of the communicating parties, with their ternary cryptographic tables, and native ternary cryptographic protocols. Public, and private key pairs are binary streams while the core of the T-PKA is based on ternary logic. The communication between parties can occurs over untrusted channels, by exchanging dynamically generated public keys, and using legacy binary codes. The proposed ternary environment largely enhances entropy, creating an additional level of cyber-protection, which can preclude malware and viruses based on binary codes to contaminate the architecture.**

Keywords— Ternary computing; public key infrastructure (PKI); authentication; cryptography; cybersecurity; physical unclonable function (PUF); memristors; ternary states.

I. Introduction – Background information

1.1 Binary versus ternary computing

In 1840 Thomas Fowler, a British mathematician, created a ternary calculating machine from wood [1-2]. Since 1958 Russia has developed, and built several generations of ternary computers, starting with the Setun. In the USA, ternary computing efforts also started in the 50's, where significant progress was made in the development of ternary arithmetic, and microelectronic components for ternary computers. Ternary computing has several fundamental advantages over binary computing.

Higher data throughput. A data stream of N bits has 2^N variables, while a data stream of N trits has 3^N variables. For example, if N=12, a binary stream has 4,096 values while a ternary stream has 531,441 values. This can result in much higher computing power, and can increase the "entropy" of crypto-systems for the same data stream length. This is a very desirable attribute for cryptography, and cryptosystems.

Access to additional instructions. A balanced ternary arithmetic is based on the three states (-, 0, +), with the "-" replacing the binary "0", and the "+ replacing the binary "1", and the additional ternary state "0" representing a fuzzy state [3]. For example, the (0, 1) in binary logic correspond to (-, +) in balanced ternary logic. Additional arithmetic instructions using the ternary state are suggested, giving an additional degree of freedom in developing algorithms. For example, there are three possible NOT gates in ternary logic: UNOT uncertainty to NOT (-0+ ➔ +0-), FNOT false to NOT (-0+ ➔ +--), and TNOT truth to NOT (-0+ ➔ ++-). Therefore, customized native ternary instruction sets can be developed to secure cryptosystems operating in a native ternary computing environment.

Back-compatibility with legacy binary codes. Binary Boolean logic can be considered as a subset of ternary Boolean logic by ignoring the instructions related to fuzzy states, so it is possible for ternary computers to directly execute some native binary codes. This is a positive attribute for information assurance, because legacy cryptographic codes such as AES (Advanced Encryption System), and DES (Data Encryption System) can be executed on ternary computers having a reduced instruction set. The performance is expected to be slightly lower, because the circuitry dedicated to the fuzzy states will then be idle, which is not a critical factor. However, modulo 2 binary arithmetic, and modulo 3 ternary arithmetic are distinct. For example, a half adder in binary arithmetic is a XOR gate for sum, and AND gate for carry, while the sum and carry in ternary arithmetic are not Boolean gates. Mathematic intensive cryptography such as RSA, and ECC requires ternary arithmetic to run on ternary engines.

Blocking malware and viruses. The lack of back-compatibility with legacy codes can be considered as an attribute for ternary computing. Malwares and virus written with binary codes cannot run directly on ternary engines. Codes written with binary instructions have to be recompiled to be executed in ternary engines, and this with the knowledge of the customized instruction set of the engines. As recompilations are necessary, the need to know these specific additional ternary instructions can prevent third parties from designing effective ternary malware and viruses.

Multi-functional units. Native general-purpose ternary computer is expected to execute efficiently ternary codes, and to execute poorly legacy binary codes. Recently it was suggested [4] to move away from the bottle neck of a monolithic ALU architecture in favor of multiple functional units, one of these units being a crypto-processor dedicated to security. We are suggesting to extend this heterogeneous

architectural concept, with one functional unit being a reduced instruction set computing (RISC) engine that executes only native ternary codes, for security purpose, see Fig.1.

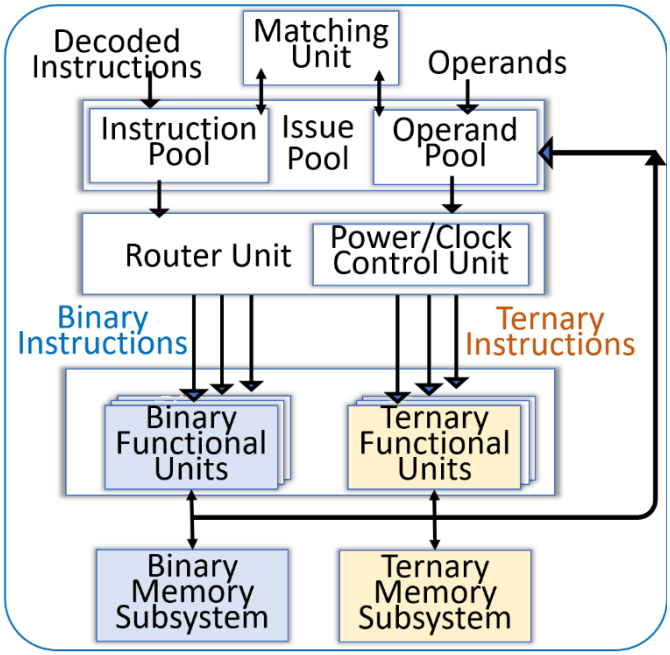

Fig.1 Downstream portion of processor datapaths shown ref [3]. One functional unit is a ternary cryptoprocessor engine.

1.2 Hardware consideration for ternary computing

Ternary logic designed with traditional CMOS technologies using a dual power supply, -1Volt/+1Volt, can execute native logic ternary codes with the -/0/+ states [5-7]. Elementary logic gates AND, OR, NOT, and XOR were developed, and documented using ternary logic. These logic gates have truth tables with 9 cases versus the 4 cases available in binary logic. Therefore additional logic functions that exist only with ternary logic were suggested. The circuitry of these ternary gates is about twice at complex as the circuitry needed for binary logic.

Ternary memories. Ternary content addressable memories (TCAM), that are widely used in routers to handle IP packets, store three states "0", "1", and "don't care". Technically TCAMs are not native ternary memories because the storage of the ternary states is done by doubling the number of SRAM cells. SRAM cells can be directly designed with ternary states, as suggested ref [8-9], for the design of TCAMs, and ternary caches. Other mainstream memory technologies that can directly store ternary states in their cells are:

- Existing multilevel NAND flash technologies are based on multiple threshold voltages to store ternary, or quaternary bits. This technology is thereby able to store more information per chip, reducing the cost per stored giga-bit. The state machines convert the ternary or quaternary information into binary information, making the flash devices look like binary memories.

- Existing DRAM memories have a design that is also naturally ternary. Q electric charges are trapped in a cell to store a "1" state, there is no charge to store a "0" state, and Q/2 charges on the reference cells. As the charges of the cells within the memory array slowly leak, the sensing element, during "read" mode, compares the remaining charges to the charge left in the reference cell. The design of a ternary DRAM device is based on trapping 0, Q/2, or Q charges on the arrays, as well as Q/4 and 3Q/4 on the reference cells. Such a design has lower design margins, and needs to be compensated with a more accurate sensing element.

- The concept of ternary Memristors, and ReRAMs has been suggested, ref [10], to also increase the bit capacity of the memory arrays. One of the methods is to create three different levels of resistivity during the programming cycle. This is done by adding more filaments between the electrodes of each cell, or by enlarging the cross-section of the existing filaments. The three states are then defined by three resistances, typically in the 1KΩ range for the "-" state, 10KΩ for the "0", and 25KΩ for the "+" state.

- Native ternary memories have been suggested, ref [11], with MRAM. The giant magneto-resistive (GMR) structure uses a stack of two antiferro-magnetic layers, and two tunnel oxides sandwiching a ferro magnetic layer, and a self-reference design. The native ternary states have thereby three different level of resistivities. A carbon nanotubes ternary memory is shown ref. [12].

1.3 Crypto-systems with ternary states

In this section, examples of prior research work on crypto-systems based on ternary states are presented.

Multifactor authentication. In the method presented in ref [13], see Fig.2, a giant ternary key is generated as a combination of multiple authentication keys such as passwords, pin codes, PUFs challenges, or other binary data streams. The longest key is kept as the original binary stream (ex: ++--+-+---++-+-+). The following keys (ex 3451 and 7237) are first converted into a decimal data stream, and used to insert ternary states inside the original binary stream with an edit distance algorithm. The giant ternary key (ex: ++-0-+0+00---0++0-0+-0+) is stored in memory for future authentication. The method reduces the efficiency of sequential attacks because it eliminates partial authentications with a single factor; the entropy of a giant ternary key is much higher than the entropy of separate binary keys combined.

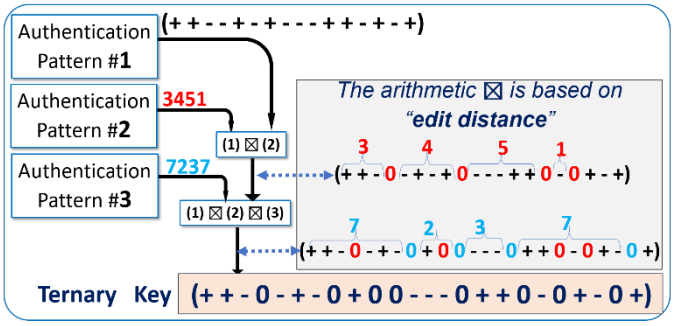

Fig.2: Generation of giant ternary keys from multiple authentication patterns.

Ternary PUFs and TRNG. Physically Unclonable Functions (PUF) exploit the natural variations introduced during the manufacturing of electronic components, and act as the "finger print" of the hardware. One of the limitations of PUFs is the drifts, and changes of the "finger prints" when subject to electromagnetic noise, temperature variations, and/or aging. To overcome this limitation, a method based on ternary states was suggested in ref [14-18], and tested with arrays of Memristors, see Fig.3. The cells of the arrays are fully tested in order to sort out the marginal cells, and blank them with a ternary state. The remaining cells are thereby reliable when tested again, during the authentication cycles. Conversely, the ternary cells can be used for a true random number generator (TRNG).

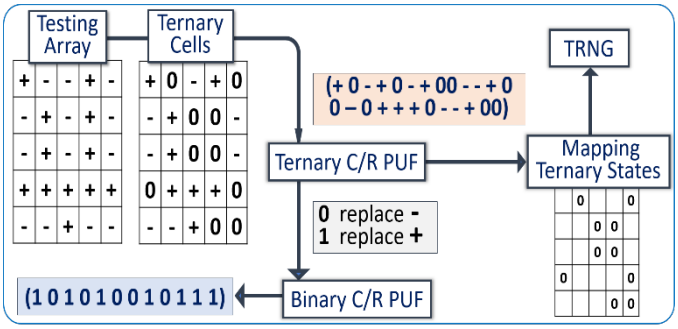

Fig.3: Method to generate ternary PUFs and TRNGs from memory arrays.

Password generation. Arrays of ternary PUFs, as described above, can be used to design password generators, called addressable PUF generators (APG) in ref [19], shown in Fig.4.

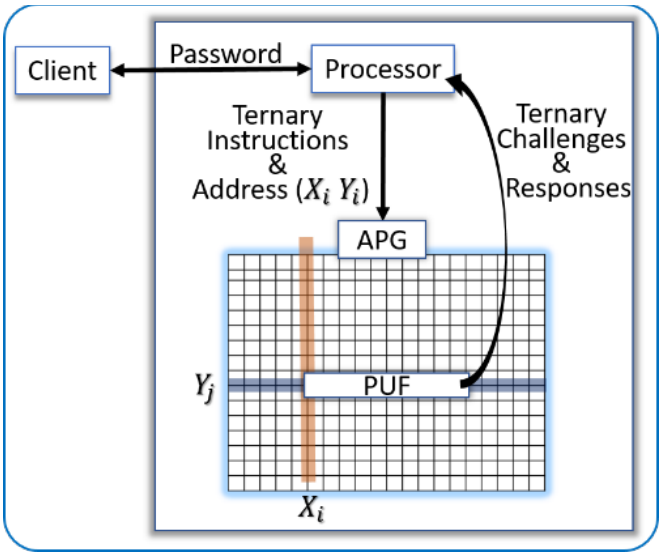

Fig.4: Password generation and authentication with arrays of ternary PUFs

During password generation, a PUF located at a particular address within the APG generates a ternary "challenge", i.e. reference pattern of the PUF, with a particular set of ternary instructions. The resulting password is the data stream combining the address, the instruction, and the challenge. During authentication the password is analyzed, the knowledge of the address is used to find the same address, with the ternary instruction a ternary "response", i.e. fresh pattern, is generated and compared to the reference challenge. The authentication is positive if the challenge matches the response. This password generation method is more secure than a look up table when combined with random numbers, hash functions, and dynamically generated passwords.

II. Ternary Public Key Infrastructure

In this section, we are presenting step by step the development of a Ternary state based Public Key exchange that is Addressable (T-PKA). The communication protocol between parties is based on legacy binary computing, while the process to generate private keys from the public keys can be implemented in native ternary computing.

2-1 Ternary cryptographic tables.

The initial step of the Ternary Public Key scheme that is Addressable (T-PKA), also called personalization, is based on the generation by the secure Server of the network, of one cryptographic table per client device, with Ternary True Random Numbers (T-TRN) of trits, and this in a fully secure environment [20]. In this paper, we are using balanced ternary logic with each trit being either a "-" state, a "0" state, or a "+" state. For example, each table can have the format of 256 rows, and 256 columns, for a total bit density of 64Ktrits, see Fig.5.

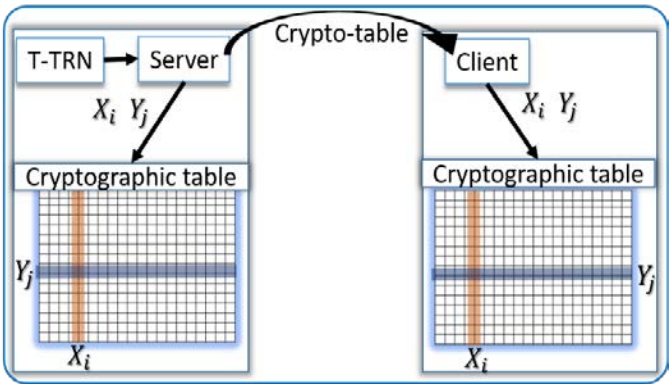

Fig.5: The server generates a crypto-table with ternary numbers for each client device.

During personalization, one cryptographic table is securely downloaded to each connected device, which could be a terminal device, a secure microcontroller, or a smartcard. These devices have secure non-volatile memories embedded in them to store their cryptographic tables, and all crypto-tables are securely kept in the server. The personalization, i.e. data generation, and transfer has to be done once for each connected device, thereby creating a dedicated, and highly secure environment. With this basic scheme, it is assumed that the non-volatile memories of the client devices are secure, as it is assumed in traditional public key infrastructure (PKI). Rather than storing the "private keys", the basic T-PKA scheme is based on the storage of crypto-tables. As discussed below, crypto-tables can be replaced by Addressable arrays of Physical unclonable function Generators (APG), which are more secure.

2.2 Public Key Generation

In this scheme, we are calling the ***public key*** of the T-PKA the information needed to generate a particular ***private key*** from the cryptographic table. As it is a standard practice with other PKI [21-25], the public key is openly shared, in a communication channel that is based on binary logic, and is assumed to be highly unsecure. In this scheme, only the server and the client device can independently generate a binary private key. As done with Elliptic Curve Cryptography (ECC), [26-32], or Quantum Key Distribution (QKD), T-PKA is a key exchange scheme. The private keys after key exchange are used to encrypt and decrypt messages, and perform authentication cycles, establishing a secure communication environment between server, and client.

Our initial implementation is based on AES-256 (Advanced Encryption Systems), however alternate methods such as polymorphic encryption need to be considered. A T-PKA scheme is only as good as the encryption method used in association with the key exchange.

In its basic form, the public key of T-PKA is a particular address $\mathbf{A_{i,j1}}$ **= {Xi, Yj}** in the crypto-table, see Fig.6.

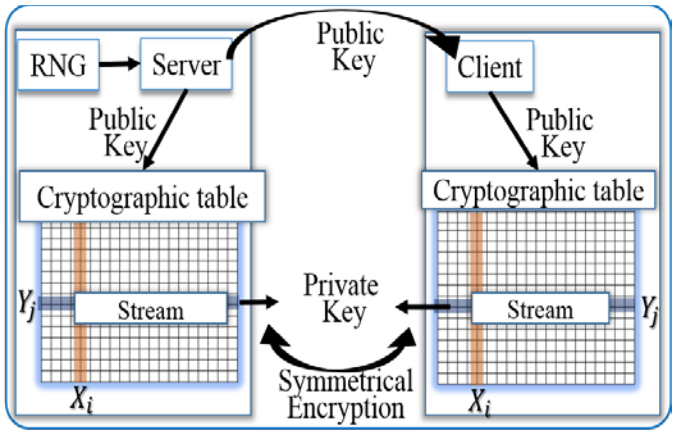

Fig.6: A public key is the set of instructions describing how to find a particular private key in the table. The encryption is based on a symmetrical scheme which use the private key.

The private key can be the stream $\mathbf{Pr_{i,j1}=C_{i,j1}=\{C_{i,j1}^{1},\ldots,C_{i,j1}^{k}\}}$ of k bits extracted from the stream of trits that are located in the cryptographic table following the address $\mathbf{A_{i,j1}}$. The "0" states are ignored, the "-"s are converted into binary "0"s, and the "+"s are converted into binary "1"s. The 256 bits needed for the private key can be extracted from the rolling following rows in the table if the address pointed by the public key is located at the end of a row. If the address pointed by the public key is located at the bottom of the table, the rolling following rows can be located at the top section of the table in such a way that a fixed length of 256 bits is always extracted for the private key.

Only the server, and the client with the appropriate cryptographic table can generate the same private key for the T-PKA protocol, a third party without the same exact cryptographic table cannot extract the same private key from the same public key. And the public key, i.e. the address in this case, can be changed often to increase the level of security.

2.3 Hash Functions and Multi-Factor Authentication

To increase the level of security, hash functions [21-23], such as the standard hash algorithm (SHA), can be used to protect the public and private keys, with an additional password, or pin code. The hash functions are one-way cryptographic functions that convert input messages into hash digests. Even a single bit change in the input message, results in a totally different hash digest. The revised protocol is shown in Fig. 7. The password can be a data stream of any length, and any origin such as a pin code, or a biometric print, i.e. finger print, vein, or retina. This password has to be known by both parties, as part of this protocol. With such an architecture, a third party should not be able to directly extract the address $\mathbf{A_{i,j1}}$ **= {Xi, Yj}** in the crypto-table from the public key. The objective is to make the knowledge of the public key pointless without knowing the password used in the hash function.

A binary random number $\mathbf{T_{i,j1}}$, generated by the Server side becomes the new public key. This random number $\mathbf{T_{i,j1}}$ generates the address $\mathbf{A_{i,j1}=\{X_i, Y_j\}}$ from the hash function and the password, as shown in Fig.7.

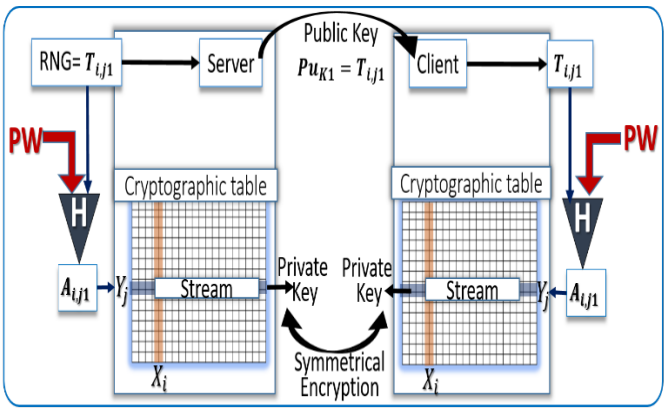

Fig.7: The hash function converts the random number $T_{i,j1}$ (the public key), and the password into the address $A_{i,j1}$ allowing the generation the private key.

If for example the ternary cryptographic table is a 256 x256 array, the first 8 digits of the hash message can be the address of the column $\mathbf{X_i}$, and the next 8 digits can be the address of row $\mathbf{Y_j}$. The private key, $\mathbf{Pr_{K1}= D_{i,j1} =\{D^{1}_{i,j1}, \ldots, D^{k}_{i,j1}\}}$, is then the stream of ***k*** binary bits located in the cryptographic table pass address $\mathbf{A_{i,j}=\{X_i, Y_j\}}$ after skipping the ternary "0", and converting the ternary "-"s and "+"s into binary "0"s and "1"s . The public key transmitted is $\mathbf{Pu_{K1}=\{T_{i,j1}\}}$. User A can generate again the address $\mathbf{A_{i,j1}=\{X_i, Y_j\}}$ with the same hash function, the same password and $\mathbf{T_{i,j1}}$, thereby the same private key $\mathbf{Pr_{K1}= D_{i,j1}}$ from the same cryptographic table. With this password protection method, a hacker cannot determine the address $\mathbf{A_{i,j}}$ from the public key $\mathbf{Pu_{K1}=\{T_{i,j1}\}}$, even if the hacker was able to uncover the cryptographic table. If a malicious party takes possession of the user's terminal device, the knowledge of the public key alone will be useless unless the malicious party also takes possession of the password. Additional levels of protection through a multi-factor authentication can be added such as biometric methods to authenticate the user. In this method, the random number $\mathbf{T_{i,j1}}$ can be dynamically changed to a different number $\mathbf{T_{i,j2}}$

prior of the following communication between parties, resulting in a different public key $\mathbf{Pu_{K2}}=\{\mathbf{T_{i,j2}}\}$, a different address $\mathbf{A_{i,j2}}$, and different private key $\mathbf{Pr_{K2}}=\mathbf{D_{i,j2}}$, thereby enhancing security.

To increase the level of protection of the scheme, the password PW used to feed the hash function can be the result of multiple levels of independent factors combined to form a giant cryptographic key of variable length [13].

2.4 Ternary states to mask the private keys

Private keys can be further protected by adding a masking pattern $\mathbf{I_{i,j1}}$ to the public key [x]: $\mathbf{Pu_{K1}}=\{\mathbf{T_{i,j1}},\mathbf{I_{i,j1}}\}$. The masking pattern $\mathbf{I_{i,j1}}$ is used to extract a binary data stream, i.e. the private key, from the cryptographic table which contains trits. The objective of this masking operation is to largely increase the number of possible private keys from the cryptographic table [33]. A mask $\mathbf{I_{i,j1}}=\{\mathbf{I^1_{i,j1}},\ldots,\mathbf{I^m_{i,j1}}\}$ having the same length $\boldsymbol{m}$ as the data stream $\mathbf{C_{i,j1}}=\{\mathbf{C^1_{i,j1}},\ldots,\mathbf{C^m_{i,j1}}\}$ that is generated with the cryptographic table past the address $\mathbf{A_{i,j}}$, and to mask all the ternary "0"s (in **blue** in the example) of the data stream, representing **t** trits, as well as replacing the ternary "-"s and "+"s with binary "0"s and "1"s.

For example the trits of $\mathbf{C_{i,j1}}$ facing a 1 in $\mathbf{I_{i,j1}}$ are masked . Additionally, an arbitrary number **h** of trits of $\mathbf{C_{i,j1}}$ are also masked with a second random number that is part of $\mathbf{I_{i,j}}$. see the **red** bits in the example bellow. This results in a new private key $\mathbf{Pr_{K1}}=\{\mathbf{Pr^1_{K1}},\ldots,\mathbf{Pr^k_{K1}}\}$ of length **k,** with **k=m-t-h**: Ternary data stream $\mathbf{C_{i,j1}}$ is extracted from the cryptographic table, the asking pattern $\mathbf{I_{i,j1}}$ is extracted from the public key **PrK1 = (0111000011010100)** :

Ternary stream $C_{i,j1}$	- - + 0 0 + + 0 + - + 0 - 0 0 0 - + - + 0 + 0 - - 0 0 + - 0 + + - 0 - +
Masking pattern $I_{i,j1}$	0 1 0 1 1 1 0 1 0 0 0 1 1 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 1 0 1 0 1 0 1
Private key Pr_{K1}	0 . 1 . . . 1 . 1 0 . . 0 . . . 0 . 0 1 . 1 . . 0 . . 1 0 . 1 . 0 . 0 .

The total number of possible private key combinations from a particular data stream $\mathbf{C_{i,j1}}$ of length **m** is increased by the factor $\binom{m-t}{h}$. This represents the number of possible combinations to insert the additional and arbitrary number of **h** "1s" in the stream of $\mathbf{I_{i,j1}}$ having **m - t** "0s" left, after **t** bits were masked "1" to eliminate the ternary trits "0". For example, if the size of the data stream $\mathbf{C_{i,j1}}$ extracted from the crypto-table is **m**=512, the number of "0" trits to be masked is **t**=182, and the number of additional blanking is **h**=74, the size of the private key is **k** = **m - t - h** = 258 bits, and the number of possible combinations is:

$$\binom{512-182}{74} \approx \mathbf{1\ 10^{75}}$$

The scheme is enhanced by encrypting $\mathbf{I_{i,j1}}$, for example by using the hash digest of section E as a cryptographic key, and AES.

Value of the use of ternary states in the crypto-tables. The use of ternary states increases the number of possible combinations of ternary random numbers that can be generated and downloaded in the cryptographic table, i.e. entropy, from $\mathbf{2^N}$ to $\mathbf{3^N}$, with **N** being the size of the table.

This also increases the number of possible combinations, in the data stream $\mathbf{C_{i,j1}}$ of **m** trits extracted at the address $\mathbf{A_{i,j1}}$ by the factor $\mathbf{3^m/2^m} = \mathbf{(1.5)^m}$. If **m**= 512, this factor equal:

$$\mathbf{(1.5)^{512} \approx 1.4\ 10^{90}}$$

The generation of the binary data stream $\mathbf{I_{i,j1}}=\{\mathbf{I^1_{i,j1}},\ldots,\mathbf{I^m_{i,j1}}\}$ has to be such that the ternary states "0" of $\mathbf{C_{i,j1}}$ need to be masked to result in a private key $\mathbf{Pr_{K1}}=\{\mathbf{Pr^1_{K1}},\ldots,\mathbf{Pr^k_{K1}}\}$ which is a binary data stream compatible with a legacy symmetrical encryption scheme such as AES. Both private, and public keys remain binary data streams, so the communication protocol between parties are back compatible with legacy infrastructure. A third party should not be statistically able to generate a working public key in man-in-the-middle attacks (pretending to be a legitimate server [21-23]) without the knowledge of the location of the ternary states "0" in the stream $\mathbf{C_{i,j1}}$. A randomly generated public key will most likely fail to mask the ternary states of the cryptographic table, and will result in a ternary private key that cannot effectively be used in a symmetrical encryption scheme. In the example shown above (**m**=512, **t**=182, **k**=258), the probability to randomly find 256 bits out of 512 that have no ternary "0" state is given by:

$$\mathbf{P} = \left(\frac{330}{512}\right)\left(\frac{329}{511}\right)\ \ldots\ \left(\frac{74}{256}\right) = \frac{330!\ 255!}{512!\ 73!} \approx \mathbf{2.14\ 10^{-78}}$$

2.5 Schemes to enhance entropy

Multiple addresses from the message digest: The message digest generated by a commercial hash function such as SHA-2 or SHA-3 can be 512-bit long, or more. Multiple addresses of the cryptographic table of the PKA can be extracted from each message digest. For example, as shown Fig.8, when the size of the cryptographic table is 256 x 256, it is possible to extract 32 different addresses from a 512-bit long message digest:

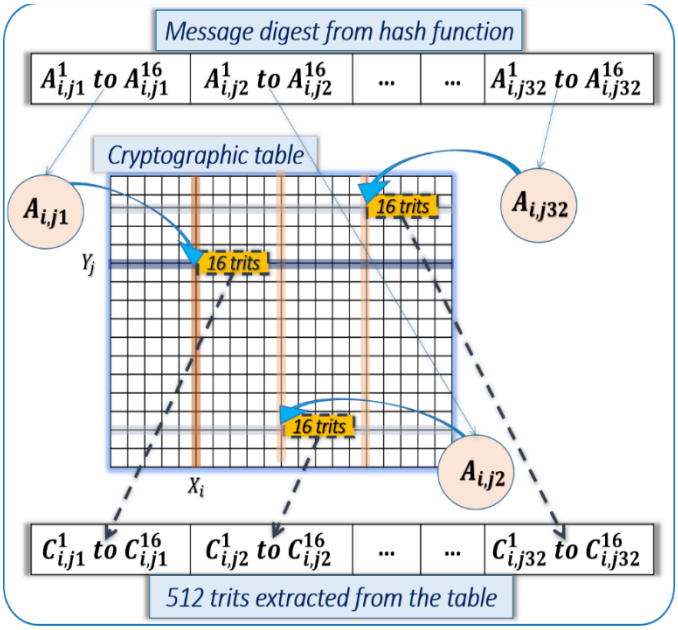

Fig.8: 32 different addresses from the hash digest to extract 512 trits

$\mathbf{A_{i,j1}}$ can be extracted from the first 16 bits of the message digest, $\mathbf{A_{i,j2}}$ from the next 16bits, all the way to $\mathbf{A_{i,j32}}$ from the last 32 bits. To extract $\mathbf{C_{i,j}}$, and get 512 trits, it is then possible to read the 16 bits following each of the 32 addresses. If the cryptographic table contains true random numbers, there are 2^{16} possible combination for each of the 32 addresses; if each address can be picked anywhere in the cryptographic table, the total number of possible configurations is:

$$(2^{16})^{32} = 2^{512} \approx 1.34\ 10^{154}$$

The number of possible configurations is slightly reduced when it is assumed that the 32 addresses cannot not overlap with each other:

$$\prod_{i=0}^{i=31}(2^{16} - 32i) \approx 1.08\ 10^{154}$$

The example discussed above of a 256 x 256 table, with a 512-bit long message digest can be generalized in tables of different sizes, and message digests shorter or longer. Cryptographic tables of smaller sizes than 256 x 256, and the same 512-bit long message digest could be segmented in more than 32 addresses. For example, a 64 x 64 cryptographic table can be segmented into 42 addresses, a 1024 x 1024 table can be segmented into 25 addresses. The example discussed above of a 256 x 256 table, with a 512-bit long message digest can be generalized to tables of different sizes, and message digests shorter or longer. Cryptographic tables of smaller sizes than 256 x 256, and the same 512-bit long message digest could be segmented in more than 32 addresses. For example, a 64 x 64 cryptographic table can be segmented into 42 addresses, a 1024 x 1024 table can be segmented into 25 addresses.

Use information stored in the cryptographic table. A different method to increase the possible combinations is to read the few trits located around the address $\mathbf{A_{i,jq}}$, and to use the information to access a set of instructions on how to generate the ternary stream $\mathbf{C_{i,jq}}$. In the example shown in Fig.5, the trits located at location $\mathbf{A_{i,jq}}$, and the one located just after, are read, a digital number from 1 to 9 can be extracted. This number, for example 7 in Fig.9, can be used to as instruction 7 to generate the trits $\mathbf{C_{i,jq}}=\{\mathbf{C_{i,jq}}^1, \ldots, \mathbf{C_{i,jq}}^{16}\}$ from the cryptographic table.

As an example of "instruction", the spacing to extract the trits following the location $A_{i,jq}$ can be edited, or spaced, with the number read in the pair of cells. In the example shown Fig.9, a "+-" at address $A_{i,jq}$ triggers the selection of the trits stored in every 7 cells in the cryptographic table. Each different address in the cryptographic table has different pairs of trits, different instructions, and will therefore space differently the generation of their streams of trits. With such an edit spacing method, there are 9 different ways to extract 16 trits located past the different 32 addresses. The number of possible configurations is then increased by:

$$(9)^{32} = 3.4\ 10^{30}$$

This method can further increase the number of configurations by reading longer streams of trits following each address. Reading 8 trits will yield the following number of configurations:

$$(8^3)^{32} \approx 5\ 10^{86}$$

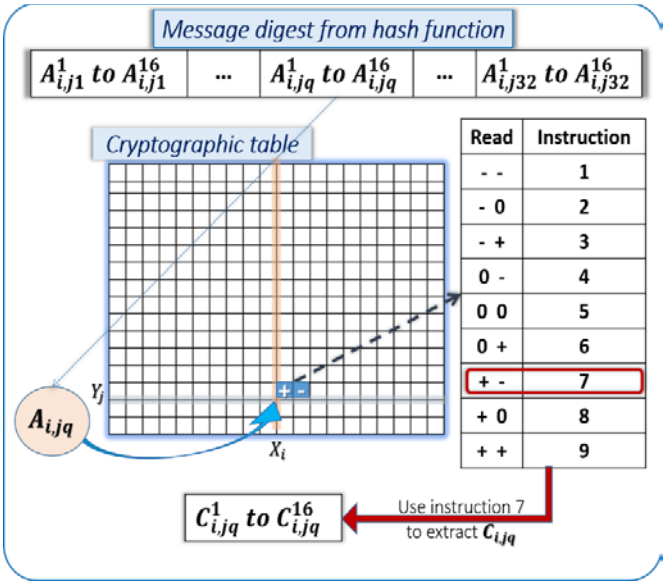

Fig.9: 32 different instructions from the crypto-table to extract 512 trits from 32 different addresses.

General instruction set. The instruction "edit spacing" can be replaced by any set of instructions needed to compute, and extract the trits following the location $A_{i,jq}$, with each different address in the cryptographic table having different instructions.

Mitigation of frequency analysis. Frequency analysis is a generic method to break cryptographic protocols that handle streams of bits of constant length. Block ciphers like AES usually encrypt blocks of 128 bits at each operation. The encryption of long plain text with thousands of words, with millions of 128 blocks, could be exposed to frequency analysis. The combination of the two methods described in section 2.1, and 2.2 can be effective to mitigate such attacks. For example, each of the 32 addresses shown Fig.1 can be used to extract chunks of 16 trits with different spacing, as shown Fig.2, and are, therefore not following a predictable sequence. The reading of the trits can vary address to address resulting in non-repetitive patterns.

The information is coming from the table itself, not the public key, this further increases the number of possible pairs of public/private key combinations. The ternary information extracted at each address can be used in many other ways to protect the scheme from frequency analysis. For example, the number 0 to 8 read on the two trits located around $\mathbf{A_{i,j1}}$ can be used to vary the length of the chunk of trits extracted; that with a 0 this length is 16, with 1 it is 17, with 2 it is 18, with 3 it is 19, and so one. The total number of chunks to extract 512 trits does not have to be 32 anymore for each public key.

2.6 Use of Physical Unclonable Functions

Arrays of ternary addressable PUF generators (APG), based on memory arrays [34-37], can generate ternary challenges and responses to replace the ternary random numbers described in section 2.1, that are filling the lock up tables, as shown Fig. 6. During initial set-up, or personalization, the APGs located at the client side securely generates look up tables of PUF ternary

challenges that are downloaded in the server in a highly secure environment. The public/private key scheme is constructed in a similar way to the one presented above, in Fig.10.

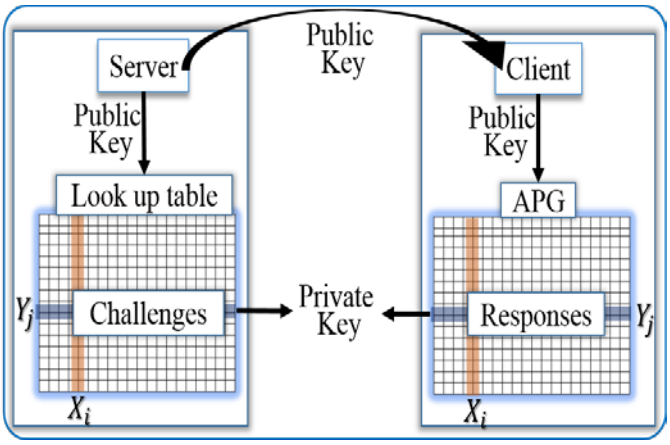

Fig.10: The T-PKA scheme is based on addressable ternary PUF arrays (APG). The private keys are based on ternary challenges or responses, not ternary random numbers.

During this personalization operation, the ternary data stream $\mathbf{C_{i,j1}}=\{\mathbf{C^1_{i,j1}},\ldots,\mathbf{C^m_{i,j1}}\}$ located at the address $\mathbf{A_{i,j1}}$ on the server side, has been replaced by a data stream of ternary challenges $\mathbf{Ch_{i,j1}}=\{\mathbf{Ch^1_{i,j1}},\ldots,\mathbf{Ch^m_{i,j1}}\}$ which will be used to generate the private key $\mathbf{Pr_{K1}}$ during the subsequent authentication cycles. On the client side, at each authentication cycle, the AGP is queried again with the public key, PUF responses $\mathbf{Re_{i,j1}}=\{\mathbf{Re^1_{i,j1}},\ldots,\mathbf{Re^m_{i,j1}}\}$ are generated at the address $\mathbf{A_{i,j1}}$ for the purpose of private key generation $\mathbf{Pr_{K1}}$. If the client device is "lost to the enemy", the arrays of PUFs cannot be directly read, thereby enhancing security. APG's can also be used separately on the server side for additional protection at the level of password management.

One noticeable difficulty in the use of PUFs, instead of a ternary random number, is the need to incorporate error correcting methods to be sure that both private keys are absolutely identical. The use of arrays of addressable PUFs, as described in this section, assume that the private keys generated independently by both the server, and the client devices are exactly the same. A single bit mismatch between the two private keys can totally derail the symmetrical encryption scheme. However, there is a definite possibility that a few bits of the PUF responses generated in the field will differ from the bits of the PUF challenge stored in the server. This difference can be caused by variations of the physical elements due to temperature changes, voltage drifts, aging, noisy measurements, or electromagnetic radiations. For this purpose, the schemes with PUFs need to integrate error correction methods in the implementation. It will ensure the exact reconstruction of a private key in the field. We can use BCH error correction scheme to recover the PUF output bits in the field [38-40].

As shown in Fig. 11, the input of an BCH encoder is a random number. The output of the encoder is XORed with the PUF response. In the reproduction stage, a noisy PUF response is XORed again with the helper data and then a BCH decoder is used to recover an error free Key. The helper data can be added to the public key, and encrypted to reduce exposure to a third party. For example, a very strong T-PKA implementation can be based on public keys of 1,536 bits. The first 512 bits represent the random number $T_{i,j1}$ needed to feed the hash function, find $A_{i,j1}$, extract $Ch_{i,j1}$ and the server side, and Re_{ij1} on the client side. The following 512 bits represent the helper data needed to correct the ternary responses, in such a way that both streams are identical. The helper can be encrypted with the massage digest of the hash function. The third 512 bits represent the masking data $I_{i,j1}$ needed to extract the private keys independently, by all parties.

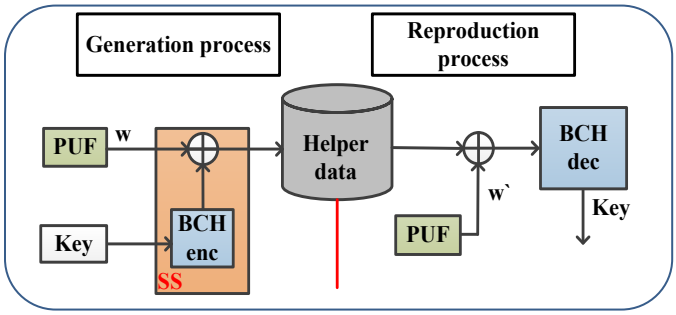

Fig.11: Error Correction scheme for PUF

III. Implementation of the T-PKA

We implemented a server-client T-PKA scheme on a PC environment. The algorithms for public-private key exchange are described in this section.

3.1 Key generation by the server

The block diagram describing the algorithms for key generation at the server side is shown below, Fig.12.

- The first step on the server side is the generation of the binary random number $\mathbf{T_{i,j1}}$, for example 512 bits. A new number can be generated at each authentication cycle between the server and the client.
- The second step is to use the hash function, and a password to generate the message digest $\mathbf{A_{i,j1}}$. Typically, SHA-2 and SHA-3 can generate a message digest of 512 bits. In our implementation we are simply XORing $\mathbf{T_{i,j1}}$, and the password to feed the hash function.
- The third step is to extract the stream of trits from the crypto-table. The 512 bits of $\mathbf{A_{i,j1}}$, are cut in 32 chunks of 16 bits. Each of the 32 chunks are pointing to one address in the crypto-table. The 16 trits following each address are extracted to generate the stream of 512 trits $\mathbf{C_{i,j1}}$. To increase entropy the trits located around each address are read, and generate instructions on how extract the 16 trits.
- The fourth step is to mask the 512 trits of $\mathbf{C_{i,j1}}$ to extract 256 bits, the private key $\mathbf{Pr_{i,j1}}$. Using the mask $\mathbf{I_{i,j1}}$, all ternary states "0" of $\mathbf{C_{i,j1}}$, are masked, as well as arbitrary additional "-" and "+" trits. This results in a stream containing only two types of trits, the "-" and the "+", that are converted in "0" and "1" bits.

- The last step is to generate the public key $\mathbf{Pu_{i,j1}}$. The first portion of the public key is the random number $\mathbf{T_{i,j1}}$. The second portion of the key is the result $\mathbf{M_{i,j1}}$, of the XORing of the message digest $\mathbf{A_{i,j1}}$, and the mask $\mathbf{I_{i,j1}}$.

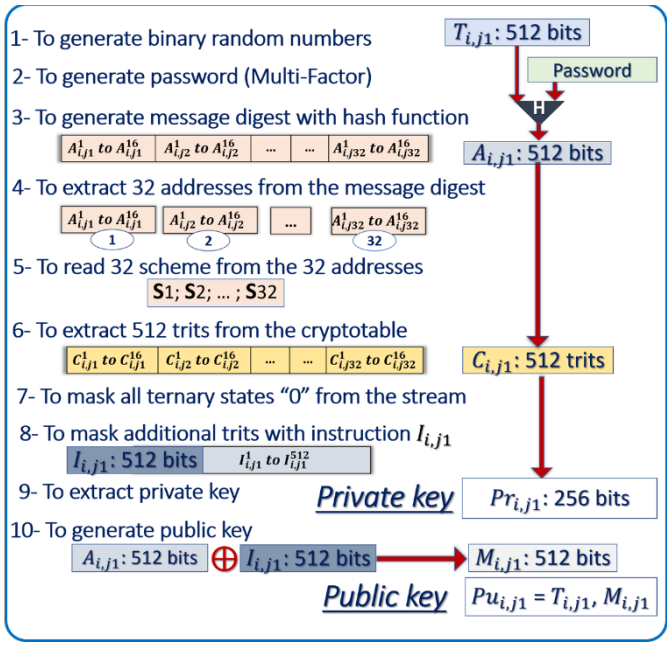

Fig.8: Diagram showing the algorithm for T-PKA key generation by the server

3-2 Key generation by the client

The block diagram describing the algorithms for key generation at the client side is shown below, Fig.9.

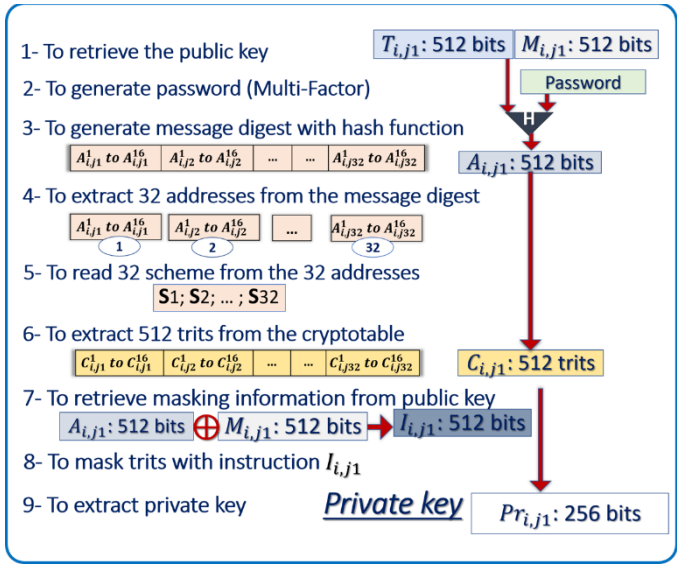

Fig.9: Diagram showing the algorithm for T-PKA key generation by the client.

- The first step is to retrieve the public key $\mathbf{Pu_{i,j1}}$ ($\mathbf{T_{i,j1}}$, $\mathbf{M_{i,j1}}$) transmitted by the server.
- The second step is to extract the same message digest $\mathbf{A_{i,j1}}$ from $\mathbf{T_{i,j1}}$, and the password.
- The third step is to find $\mathbf{I_{i,j1}}$ by XORing $\mathbf{T_{i,j1}}$ and $\mathbf{M_{i,j1}}$.
- The fourth step is to extract the stream of trits $\mathbf{C_{i,j1}}$ with $\mathbf{A_{i,j1}}$, and the crypto-table.
- The last step is to mask $\mathbf{C_{i,j1}}$ with $\mathbf{I_{i,j1}}$ to generate the private key $\mathbf{Pr_{i,j1}}$.

3-3 Entropy considerations in the implementation

The table shown below is a summary of the elements of randomness, the entropy, introduced by each factor during the implementation of the T-PKA scheme.

A- Initial step: The generation of a table with 256x256 random trits during personalization contains an extremely large level of entropy: $E_A = \mathbf{Log_2(3^{65536}) \approx 10,3868}$

The entropy due to the ternary logic is 1.5 higher.

***B- Basic scheme*:** In this scheme, only the first16 random bits of the message digest are used to find an address within the crypto-table, and generate the private key. So, the number of possible configurations has low entropy**:**

$$E_{B1} = Log_2(2^{16}) = 16$$

The private key are 256 bits long, the number of possible keys is: $E_{B2} = Log_2(2^{256}) = 256$

***C- Hashing*:** To feed the hash function, random numbers of 512-bits need to be generated, as well as 512-bit passwords: $E_{C1} = Log_2(2^{1024}) = 1024$

Message digests of 512 bits are generated, with entropy $E_{C2} = Log_2(2^{512}) = 512$, however the collision is high when only the first 16 bits are used to extract the private keys.

***D- Masking*:** The masking operation eliminates the ternary "0"s to leave behind a private key which has to be a binary stream. The number of possible ways to mask 182 memory cells out of 512 leads to the entropy $E_{D1} = 257$.

In the implementation of the T-PKA scheme we masked an arbitrary number of cells to leave behind a final private key of 256 bits, creating an additional entropy $E_{D2} = 260$.

E- Multiple addresses: The first advantage of the protocol using 32 addresses in the crypto-table from the 512-bit message digest is to exploit the entire message digest, rather than the first 16 bits. This protocol reduces the collision problem highlighted above, section C. The number of ways to find randomly 32 addresses create the entropy $E_{E1} = 513$.

Step	Factor	Number of possibilities	Entropy
A- Personalization	256 x 256 table random numbers	$3^{65,536} \approx 2^{103868}$	103868
	Ways to pick totally independent streams of 512 trits in the table	128	7
B- Basic scheme	Number of possible addresses	2^{16}	16
	Probability to blindly find a private key of 256 bits	$\approx 2^{-256}$	256
C-Hash function & PW	512 bits random number/password	$2^{512} \times 2^{512}$	1024
	512 bits message digest	2^{512}	512
D- Masking scheme	Probability to blindly mask the 182 ternary "0"s out of 512	$(\frac{330!256!}{512!73!}) \approx 2.14\ 10^{-77}$ $\approx 2^{-257}$	257
	Ways to mask 74 trits out of 330	$(\frac{512-182}{74}) \approx 1\ 10^{75}$ $\approx 2^{249}$	260
E- Multiple Addresses	Ways to pick 32 addresses	$(2^{16})^{32} \approx 1.48\ 10^{154}$ $\approx 2^{513}$	512
	Ways to adjust the space of 16 trits by reading 8 trits at 32 addresses	$(8^3)32 \approx 5\ 10^{86}$ $\approx 2^{287}$	287

To Exploit the trits located at each of the 32 addresses, and to modify the way of extracting the private keys can enhance entropy. A protocol based on the reading of 8 trits read at each of 32 addresses, can increase entropy by $E_{E2} = 287$.

IV Conclusion and Future Work

At every step, the use of ternary states at the client/server side strengthens the T-PKA scheme, while the interested parties can communicate with legacy binary cryptography. Our implementation was an iterative process aiming at the prevention of possible attacks, and the enhancement of entropy. A quasi-infinite number of public keys allows the implementation of a one-time public key protocol. The hash function with multi-factor authentication, which is a mainstream cryptographic method, is aimed at creating a barrier between the public key, and creating the ability to hide the locations of relevance in the cryptographic tables.

The obligation to know where the ternary states are located in the cryptographic table prevents a third party from randomly finding a public key that will be able to generate a private key, to query the client. The use of multiple addresses, and the reading of the ternary content of the cryptographic tables create a quasi-infinite number of possible private keys per public key.

This work is part of a wider project that we are conducting that includes the development of heterogenous binary/ternary units, native ternary coding, ternary PUFs, ternary random number generators, and multi-factor authentication with ternary keys. We are, in particular, studying how the T-PKA code can be written for the ternary computing unit.

The T-PKA key exchange protocol does not use arithmetic instructions, and therefore does not consume significant computing power. This protocol is based on shift registers, and Boolean logic, which is relatively strait forward to transfer to a native ternary computing environment.

Another area of research is related to polymorphic cryptography. In the implementation that is described in this paper the message digest is pointing toward multiple addresses within the cryptographic table with the objective of generating a single private key. This protocol can be changed to generate multiple private keys for polymorphic cryptography.

Lastly, we are also studying ways to use the polymorphic nature of T-PKA to make the hardware used by the client distinct, and constantly changing over-time.

In conclusion, ternary computing is creating an additional degree of freedom that can be extremely valuable to strengthen cybersecurity. We are not advocating the use of generic ternary computers however, in favor of a heterogeneous ternary/binary architecture. In this environment, we are considering the T-PKA as extremely promising.

Aknowledgment

The authors are thanking the Air Force Research Laboratories of Rome NY for their support, as well as the students and faculty from Northern Arizona University, in particular Christopher Philabaum, Duane Booher, Bilal Habib, Raul Chipana, Paul Flikkema, and James Palmer.

REFERENCES

[1] M.G. Nektar, D.M. Hogan, P. Vass; The ternary calculating machine of Thomas Fowler; IEEE Annals of the History of Computing, Aug 2005;

[2] A.A. Obiniyi, E.E. Absalom, K. Adako; Arithmetic Logic Design with Color Coded Ternary for Ternary Computing; ICA, vol 26 No11, July 2011;

[3] S. Ahmad, M. Alam; Balanced Ternary Logic For improving Computing; IJCSIT, 2014;

[4] P.G. Flikkema and B. Cambou; Adapting Processor Architectures for the Periphery of the IoT Nervous System; IEEE 3rd World Forum on Internet of Things (WF-IoT), December 2016;

[5] X.W. Wu; CMOS Ternary Logic Circuits; IEE Proceedings, Feb 1990

[6] P.C. Balla, A. Antoniou; Low Power Dissipation MOS Ternary Logic Family; IEEE J of Solid State Circuits, Oct 1984;

[7] Ion Profeanu; A ternary Arithmetic and Logic; WCE, June 2010;

[8] N.P. Wanjari, S.P. Hajare; VLSI Design and Implementation of Ternary Logic Gates and Ternary SRAM Cell; IJECSE, April 2013;

[9] P. Nagaraju, and all; Ternary Logic Gates and Ternary SRAM Implementation in VLSI; IJSR, Nov 2014;

[10] M. Khalid, J. Singh; Memristor based unbalanced ternary logic gates; Analog Integrated Circuits and Signal Processing, April 2016;

[11] B. Cambou; Multilevel magnetic element; US patent 8,630,112; Jan 2014;

[12] S. Lin, Y-B. Kim, F Lombardi; CNTFET-Based Design of Ternary Logic Gates and Arithmetic Circuits; IEEE Trans on Nanotechnologies; March 2011;

[13] B. Cambou; Multi-factor authentication using a combined secure pattern; US patent 9,514,292, Jul 2015;

[14] B.Cambou, and M. Orlowski; Design of PUFs with ReRAM and ternary states; CISR 2016, April 2016;

[15] B. Cambou, F. Afghah; PUF with Multi-states and Machine Learning; CryptArchi 2016;

[16] B. Cambou; PUF generating systems and related methods; US patent disclosure No: 62/204912; Aug 2015;

[17] B. Cambou; A XOR data compiler combined with PUF for TRNG; SAI/IEEE computing conference, July 2017 (accepted);

[18] B. Cambou; Data compiler for True Random Number Generation and Related Methods; NAU disclosure D2017-03, Aug 2016;

[19] B.Cambou; Encoding Ternary Data for PUF Environment; NAU disclosure D2017-11, Sept 2016;

[20] B. Cambou; Encryption Schemes with Addressable Elements; NAU disclosure D2017-21, Jan 2017.

[21] C. Paar, J. Pezl; Understanding Cryptography- A text book for students and practitioners; Spinger editions, 2011;

[22] H.X. Mel, D. Baker; Cryptography Decrypted; Addison-Wesley editions, 2001;

[23] C. P. Pfleeger, and al; Security in Computing; Fifth edition; Prentice Hall editions, 2015;

[24] Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory IT-22, 644–654 (1976).

[25] Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21(2), 120–126 (1978).

[26] Goldreich, O.: Foundations of Cryptography, vol. 1. Cambridge University Press, Cambridge (2001).

[27] Schneier, B.: Applied Cryptography: Protocols, Algorithms, and Source Code in C. John Wiley, Chichester (1996).

[28] M. Bellare and all, "Entity authentication and key distribution", Advances in Cryptology, - CRYPTO'93, Lecture Notes in Computer Science Vol. 773, D. Stinson ed, Springer-Verlag, 1994, pp. 232-249.

[29] M. Bellare and P. Rogaway; "Probably secure session key distribution– the three-party case"; STOC 1995.

[30] M. Bellare, R. Canetti and H. Krawczyk, "A modular approach to the design and analysis of authentication and key-exchange protocols"; STOC, 1998.

[31] Krawczyk, "SKEME: A Versatile Secure Key Exchange Mechanism for Internet,", Proceedings of the 1996 Internet Society Symposium on Network and Distributed System Security, pp. 114-127.

[32] NIST Recommendation for Key Management Special Publication 800-57 Part 1 Revision 4.

[33] B. Cambou, R. Chipana, B. Habib; Securing PUFs with Additional Random Ternary States; NAU disclosure D2017-19, Dec 2016;

[34] Y. Jin; Introduction to hardware security; Electronics 2015, 4, 763-784; doi:10.3390/electronics4040763;

[35] Pravin Prabhu and all; Extracting Device Fingerprints from Flash Memory by Exploiting Physical Variations; 4th international conference on Trust and trustworthy computing; June 2011;

[36] Daniel E Holcomb, and all; Power up SRAM state as an identifying Fingerprint and Source of True Random Numbers; IEEE Trans. On Computers, 2009 Vol 58, issue No09 Sept;

[37] An. Chen; Comprehensive Assessment of RRAM-based PUF for Hardware Security Applications; IEDM IEEE; 2015;

[38] N. Beckmann and all, ''Hardware-based public-key cryptography with public physically unclonable functions,'' in Information Hiding. New York, NY, USA: Springer-Verlag, 2009, pp. 206–220.

[39] B. Habib, J. Kaps and K. Gaj ."Efficient SR-Latch PUF," Proc. ISARC, 2015, Bochum, Germany, April 15-17. 2015.

[40] H. Kang, Y. Hori, T. Katashita, M. Hagiwara and K. Iwamura. "Cryptographic Key Generation from PUF Data Using Efficient Fuzzy Extractors," in Proc. ICACT, 2014, pp.23–26.