AFRL-RI-RS-TR-2019-032

BENCHMARKING TERNARY COMPUTING FOR INCREASED INFORMATION ASSURANCE

FEBRUARY 2019

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2019-032 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
JOSEPH A. CAROLI
High Performance Systems Branch
Computing & Communications Division

/ S /
GREGORY J. ZAGAR
Deputy Chief, Computing
& Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
FEBRUARY 2019	FINAL TECHNICAL REPORT	APR 2017 – JAN 2019

4. TITLE AND SUBTITLE

BENCHMARKING TERNARY COMPUTING FOR INCREASED INFORMATION ASSURANCE

5a. CONTRACT NUMBER
IN-HOUSE: R2B6

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62788F

6. AUTHOR(S)

Donald Telesca

5d. PROJECT NUMBER
T2MD

5e. TASK NUMBER
IN

5f. WORK UNIT NUMBER
HO

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/Information Directorate
Rome Research Site/RITB
525 Brooks Road
Rome NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/Information Directorate
Rome Research Site/RITB
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TR-2019-032

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. PA# 88ABW-2019-0573
Date Cleared: 11 Feb 19

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The objective of this research work was to develop a ternary based public key exchange scheme that is addressable to replace, or complement, the existing Public Key Infrastructures (PKI). The Ternary Addressable Public Key Infrastructure (TAPKI) leveraged arrays of physical unclonable functions (PUFs) and heterogeneous ternary/binary computing systems. 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 enhanced entropy, creating an additional level of cyber-protection.

15. SUBJECT TERMS

Ternary, addressable PUF generator, PUF heterogeneous computing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**DONALD TELESCA**
U	U	U	UU	21	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

TABLE OF CONTENTS

Section **Page**

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

1. SUMMARY

The objective of this research work was to develop a ternary based public key exchange scheme that is addressable to replace, or complement, the existing Public Key Infrastructures (PKI). The Ternary Addressable Public Key Infrastructure (TAPKI) leveraged arrays of physical unclonable functions (PUFs) and heterogeneous ternary/binary computing systems. Public and private key pairs are binary streams while the core of the TAPKI is based on ternary logic. The communication between parties can occur over untrusted channels, by exchanging dynamically generated public keys, and using legacy binary codes. The proposed ternary environment largely enhanced entropy, creating an additional level of cyber-protection.

While the initial benchmarking research work was considered as a compelling generic proof of concept, which validated the potential value of ternary computing for Information Assurance, the anticipated outcome of the research work was to develop the key technology modules needed to deploy ternary cryptography that can secure strategic assets.

The main objectives were the:

a) Characterization of packaged arrays of balanced ternary PUFs (-, 0, +) fabricated with memristors, or ReRAM (Resistive Random-Access Memory) arrays, in preparation for the design of multichip-modules integrated with crypto-processors;
b) Development of a multi-key Ternary Addressable Public Key Infrastructure (TAPKI) based on randomly generated ternary tables: multi-addresses, and multi-key, for one time use polymorphic cryptography;
c) Development of heterogeneous ternary RISC (Reduced Instruction Set Computer) engines, hardware/software (HW/SW) to drive this cryptography, and the development of a Field-programmable gate array (FPGA)-based emulation of these engines.

2. INTRODUCTION

Hackers and cyber-criminals continuously probe and attack legacy infrastructure. The layers of assurance that are added to our systems after-the-fact can be themselves susceptible to the same attacks. It may be the time to ask the following question: should we consider radical architectural and infrastructural changes that may disrupt the status quo to support a healthier cyber-security ecosystem through computational diversity?

Ternary computing uses trits (three primitive values of 0, 1, and 2, or balanced values of -, 0, and +) to enable denser numerical encodings, support a fuzzy state, and give the cryptographer the opportunity to introduce new ciphering methods based on hardware primitives that can provide additional security. However, general-purpose ternary computers have not been successful so far, because they are not as simple as binary computer designs for working with binary encoded data and existing binary architectures.

This work is a re-introduction of ternary computing, a technology invented 150 years ago but that has seen little practical application because it has not demonstrated advantages over binary computing in terms of computability or processing performance. Instead of applying ternary computing to processing, this work was predicated on the hypothesis that the integration of native ternary hardware with binary hardware can improve cyber security.

If ternary computing has the potential to disrupt and change the landscape of cybersecurity, we must ask a second question: can the inherent complexity and implementation costs associated with ternary computing be mitigated such that this technology can be used as a compelling fortification against malicious entities? In this work, the limitations of current security protocols were explored and the design of ternary computing units was demonstrated to take advantage of recent advances in semiconductor technology. As an example, the design of public key exchange protocols between a computer and a secure microcontroller using the Java Card OpenPlatform that takes advantage of ternary operations and representation was completed.

3. METHODS, ASSUMPTIONS, AND PROCEDURES

3.1. Ternary cryptographic tables.

The initial step of the TAPKI exchange scheme, 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[1]. 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 Figure 1.

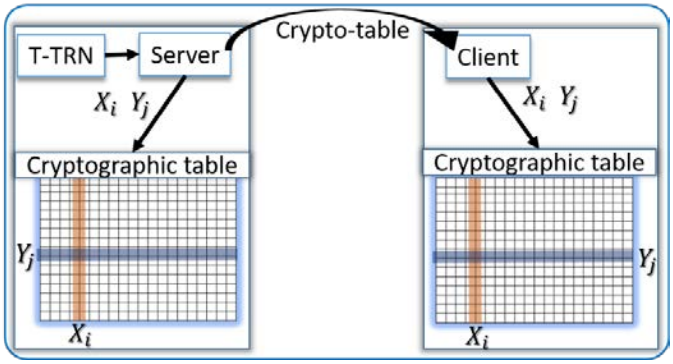

Figure 1: The server generates a crypto-table with ternary numbers for each client device.

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 TAPKI 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.

3.2. Public Key Generation

In this scheme, we are calling the ***public key*** of the TAPKI the information needed to generate a particular ***private key*** from the cryptographic table. As it is a standard practice with other PKI [2,3,4,5,6] 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.

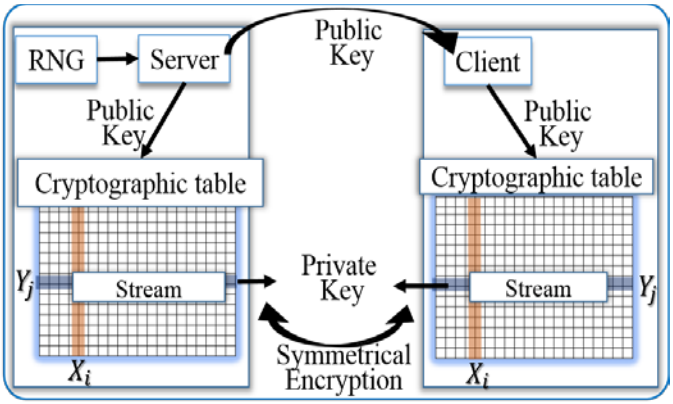

Figure 2: A public key is the set of instructions describing how to find a particular private key in the table. The encryption is based on a symmetrical scheme which use the private key.

As done with Elliptic Curve Cryptography (ECC), [26-32], or Quantum Key Distribution (QKD), TAPKI 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.

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

In its basic form, the public key of the TAPKI is a particular address $\mathbf{A_{i,j1} = \{Xi, Yj\}}$ in the crypto-table, see Figure 2. 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 TAPKI 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.

3.3. Hash Functions and Multi-Factor Authentication

To increase the level of security, hash functions, 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 Figure 3.

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 Figure 3.

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

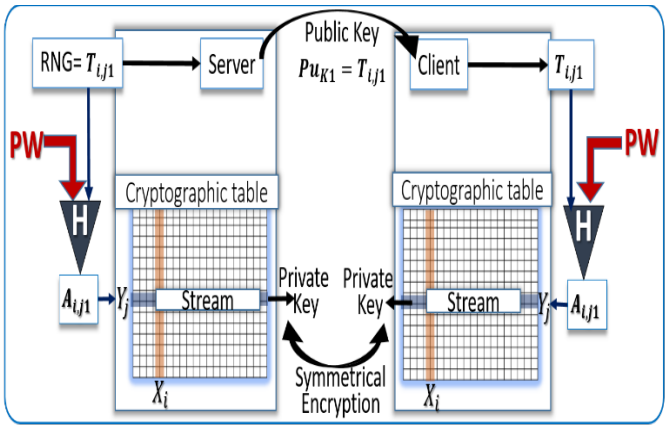

Figure 3: The hash function converts the random number Ti,j1 (the public key), and the password into the address Ai,j1 allowing the generation the private key.

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[7].

3.4. 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]: $\mathbf{Pu_{K1}=\{T_{i,j1},I_{i,j1}\}}$. 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 $\mathbf{I_{i,j1}=\{I^1_{i,j1},\ldots,I^m_{i,j1}\}}$ having the same length ***m*** as the data stream $\mathbf{C_{i,j1}=\{C^1_{i,j1},\ldots,C^m_{i,j1}\}}$ 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 $\mathbf{Pr_{K1}=\{Pr^1_{K1}, \ldots ,Pr^k_{K1}\}}$ 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 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** = 256 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.

3.5. 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 $\mathbf{3^m/2^m = (1.5)^m}$. If **m**= 512, this factor equal:

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

The generation of the binary data stream $\mathbf{I_{i,j1}=\{I^1_{i,j1}, \ldots, I^m_{i,j1}\}}$ has to be such that the ternary states "0" of $\mathbf{C_{i,j1}}$ need to be masked to result in a private key $\mathbf{Pr_{K1}=\{Pr^1_{K1}, \ldots, Pr^k_{K1}\}}$ 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**=256), the probability to randomly find 256 bits out of 512 that have no ternary "0" state is given by:

$$\mathbf{P} = \left(\frac{330}{512}\right)\left(\frac{329}{511}\right) \ \ldots \left(\frac{75}{257}\right) = \frac{330!\ 256!}{512!\ 74!} \approx \mathbf{2.11\ 10^{-78}}$$

3.6. 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-bits long, or more. Multiple addresses of the cryptographic table of the PKI can be extracted from each message digest. For example, as shown Figure 4, 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. $\mathbf{A_{i,j1}}$ can be extracted from the first 16 bits of the message digest, $\mathbf{A_{i,j2}}$ from the next 16 bits, 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 combinations 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:

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

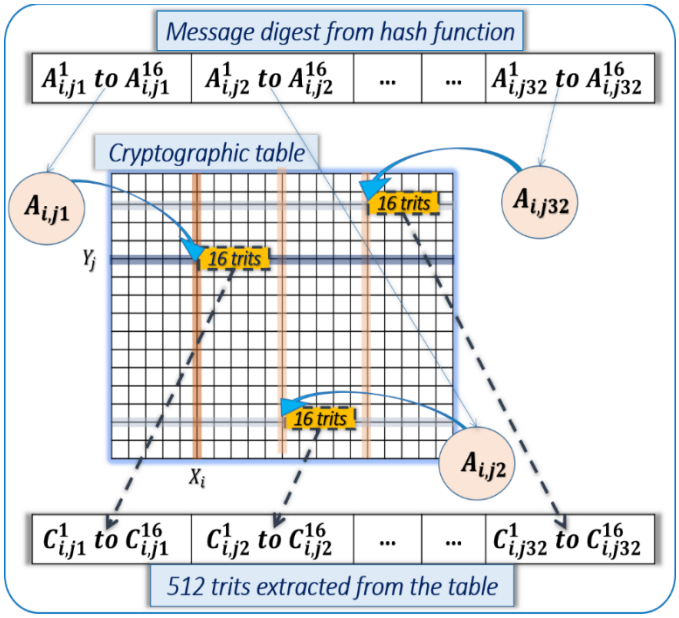

Figure 4: 32 different addresses from the hash digest to extract 512 trits

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 Figure 4, 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 Figure 5, can be used as instruction 7 to generate the trits $\mathbf{C_{i,jq}=\{C_{i,jq}^{1}, \ldots, C_{i,jq}^{16}\}}$ 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 in Figure 5, 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, and different instructions 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: $\mathbf{(9)^{32} = 3.4\ 10^{30}}$

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:

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

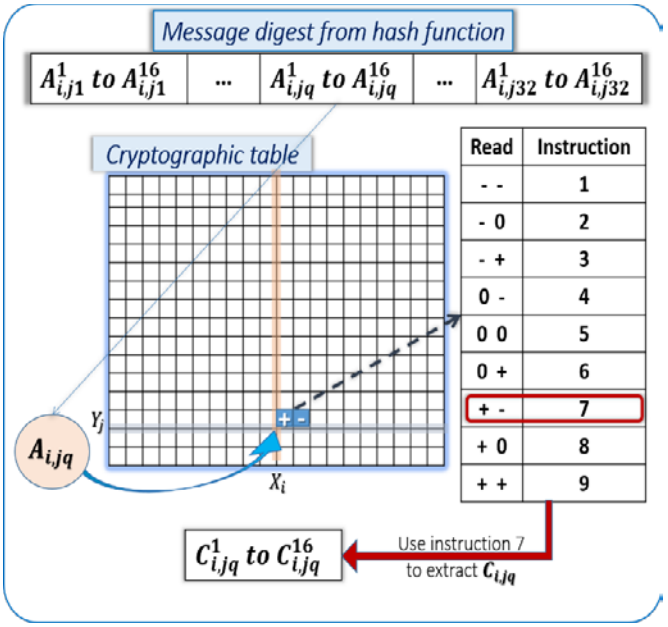

Figure 5: 32 different instructions from the crypto-table to extract 512 trits from 32 different addresses.

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 can be used to extract chunks of 16 trits with different spacing, 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.

3.7. Use of Physical Unclonable Functions

Arrays of ternary addressable PUF generators (APG), based on memory arrays [8,9,10,11] can generate ternary challenges and responses to replace the ternary random numbers described in section 2.1, that are filling the look up tables, as shown Figure 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 Figure 4.

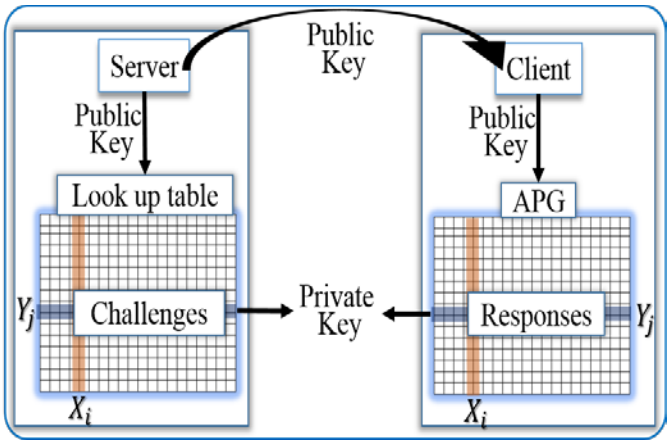

Figure 6: The TAPKI scheme is based on addressable ternary PUF arrays (APG). The private keys are based on ternary challenges or responses, not ternary random numbers.

During this personalization operation, the ternary data stream $\mathbf{C_{i,j1}=\{C^1_{i,j1},\ldots,C^m_{i,j1}\}}$ located at the address $\mathbf{A_{i,j1}}$ on the server side, has been replaced by a data stream of ternary challenges $\mathbf{Ch_{i,j1}=\{Ch^1_{i,j1},\ldots,Ch^m_{i,j1}\}}$ 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 $\mathbf{Re_{i,j1}=\{Re^1_{i,j1},\ldots, Re^m_{i,j1}\}}$ are generated at the address $\mathbf{A_{i,j1}}$ for the purpose of private key generation $\mathbf{Pr_{K1}}$. 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 (Bose, Chaudhuri, and Hocquenghem) error correction scheme to recover the PUF output bits in the field.

As shown in Figure 7, the input of a 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 TAPKI 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 message 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.

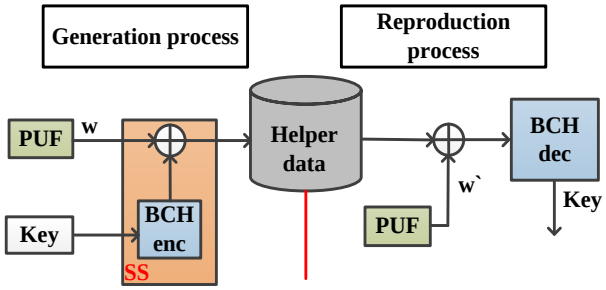

Figure 7: Error Correction scheme for PUF

3.8.Implementation of the TAPKI

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

Key generation by the server

The block diagram describing the algorithms for key generation at the server side is shown below, Figure 8.

- 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 to "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 $\mathbf{I_{i,j1}}$.

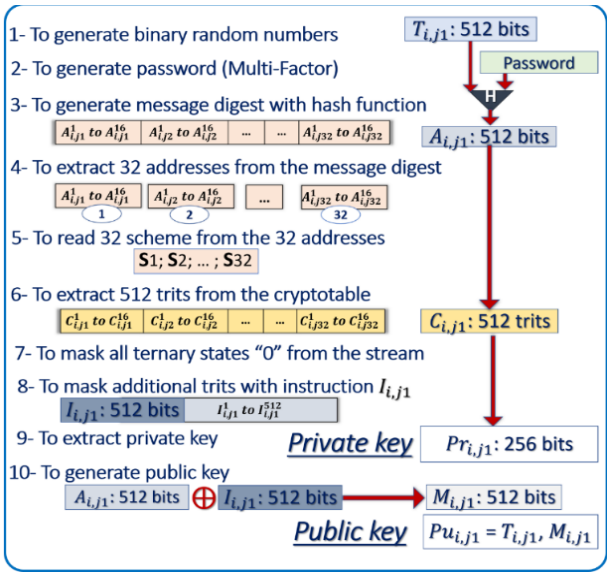

Figure 8: Diagram showing the algorithm for TAPKI key generation by the server

Key generation by the client

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

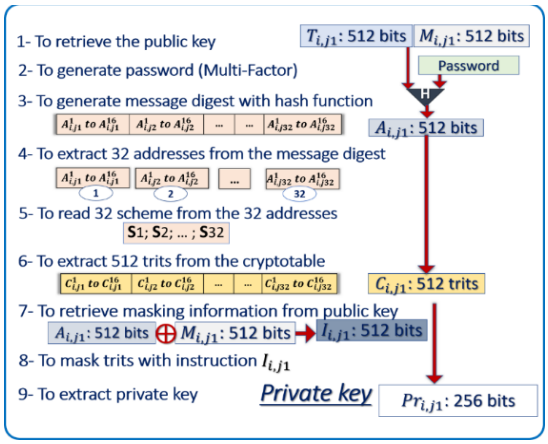

Figure 9: Diagram showing the algorithm for TAPKI key generation by the client.

- 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 $\mathbf{Pr_{i,j1}}$.

4. RESULTS AND DISCUSSION

Each factor described in this key exchange protocol was introduced to enhance the level of randomness, i.e. the entropy, and mitigate brute force attacks, and frequency analysis having the objective to extract the keys. In this section, we are quantifying the effect of each factor of TAPKI, and summarizing it in table 1 shown below.

a. ***Initial step***: The generation of a table with 256x256 random trits during personalization contains an extremely large level of entropy:

$$E_A = Log_2(3^{65536}) \approx 10,3868$$

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:

$$E_{D1} = 258.$$

In the implementation of the TAPKI scheme, we masked an arbitrary number of cells to leave behind a final private key of 256 bits, creating an additional entropy:

$$E_{D2} = 249.$$

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 in section C. The number of ways to find randomly 32 addresses create the entropy: $E_{E1} = 512$.

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

f. $E_{E2} = 288$.

TABLE 1. ENTROPY CALCUATION

Step	Summary of the elements of randomness introduced by each factor of the TAPKI scheme		
	Factor	*Number of Possibilities*	*Entropy*
a. Personalization	256 x 256 table of random numbers	$3^{65,536} \approx 2^{103868}$	103868
	Ways to pick streams of 512 trits in the table	128	7
b. Basic Scheme	Number of possible addresses	2^{16}	16
	Probability to 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 mask 182 ternary "0"'s out of 512	$(\frac{330!\ 256!}{512!\ 74!}) \approx 2.11 \times 10^{-78} \approx 2^{-258}$	258
	Ways to mask 74 trits out of 330	$\binom{512-182}{74} \approx 1 \times 10^{75} \approx 2^{249}$	249
e. Multiple addresses	Ways to pick 32 addresses	$(2^{16})^{32} \approx 1.34 \times 10^{154} \approx 2^{512}$	512
	Combinations when 8 trits are read at each of the 32 addresses	$(8^3)^{32} \approx 5 \times 10^{86} \approx 2^{288}$	288

A server-to-client TAPKI prototype was designed to evaluate the protocol, and compare it to other key exchange methods in use for PKI: RSA (Rivest–Shamir–Adleman), and ECC (Elliptical curve cryptography). The ternary random numbers needed for the crypto-tables, 64K trits, were generated and downloaded into the memory of each client device.

The example of server-to-terminal prototype shown below, use the private key generated by the TAPKI, and AES-256, to encrypt and securely transmit 2.5Mbyte Microsoft word files "inf633" between the server and the terminals. All benchmarking results were run on a MacBook Pro 3.1 GHz Intel® Core TM i7.

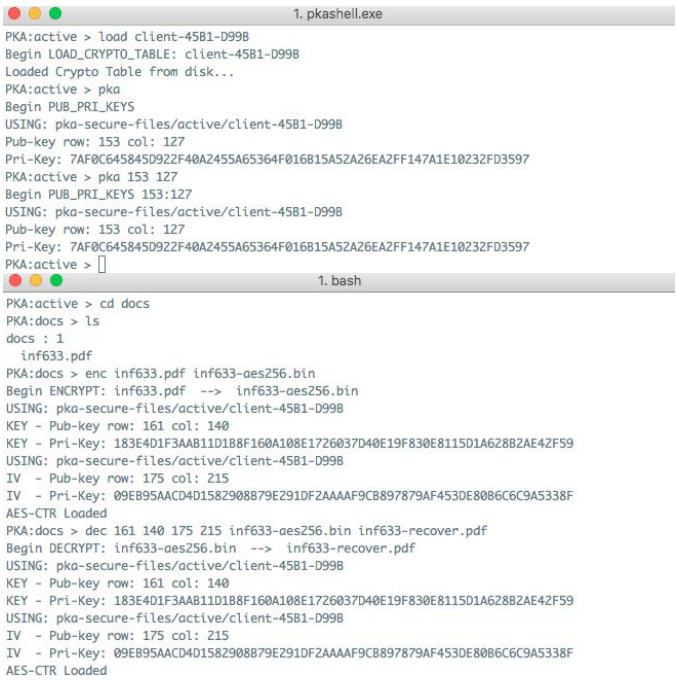

Figure 10 Example of server-to-terminal prototype

The TAPKI key exchange with hashing function, multi-factor authentication, and multi-addressing protocol takes 809 CPU (Central Processing Unit) clock cycles for 256-bit keys. Based on the experimental verification done in our computing environment, it is significantly less than the equivalent figures for both ECC (about 96,000 CPU clock cycles for $\mathscr{F}2256$) and RSA (494,000 CPU clock cycles for 3,072-bit keys) key exchange schemes for similar key strength.

5. CONCLUSION

At every step, the use of ternary states at the client/server side strengthens the TAPKI scheme, while the interested parties can communicate with legacy binary cryptography. Our implementation was an iterative process aimed 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 heterogeneous binary/ternary units, native ternary coding, ternary PUFs, ternary random number generators, and multi-factor authentication with ternary keys. Of particular importance is determining how to transition the single device PUF success that has been demonstrated into a packaged chip. The packaged chip version must be extensively tested for reliability and robustness. Once the hardware has been advanced and validated, integration into the TAPKI protocol becomes the next research step that will be addressed for final development. Finally, we are concurrently developing how the TAPKI code can be written for the ternary computing unit.

The TAPKI 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 TAPKI 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 TAPKI as extremely promising.

6. REFERENCES

[1] B. Cambou; Encryption Schemes with Addressable Elements; NAU disclosure D2017-21, Jan 2017.
[2] C. Paar, and J. Pezl; Understanding Cryptography- A text book for students and practitioners; Springer editions, 2011;
[3] H.X. Mel, and D. Baker; Cryptography Decrypted; Addison-Wesley editions, 2001;
[4] C. P. Pfleeger, et al; Security in Computing; Fifth edition; Prentice Hall editions, 2015;
[5] W. Diffie, and M. Hellman; New directions in cryptography. IEEE Transactions on Information Theory IT-22, 644–654 (1976).
[6] R. Rivest, A. Shamir, and L. Adleman; A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21(2), 120–126 (1978).
[7] B. Cambou; Multi-factor authentication using a combined secure pattern; US patent 9,514,292, Jul 2015;
[8] Y. Jin; Introduction to hardware security; Electronics 2015, 4, 763-784; doi:10.3390/electronics4040763;
[9] Pravin Prabhu et al; Extracting Device Fingerprints from Flash Memory by Exploiting Physical Variations; 4th international conference on Trust and trustworthy computing; June 2011;
[10] Daniel E Holcomb, et al; Power up SRAM state as an identifying Fingerprint and Source of True Random Numbers; IEEE Trans. On Computers, 2009 Vol 58, issue No09 Sept;
[11] An. Chen; Comprehensive Assessment of RRAM-based PUF for Hardware Security Applications; IEDM IEEE; 2015;

LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS

AES	Advanced Encryption Systems
APG	Addressable arrays of Physical unclonable function Generators
BCH	Bose, Chaudhuri, and Hocquenghem error correction
CPU	Central Processing Unit
ECC	Elliptic Curve Cryptography
FPGA	Field-Programmable Gate Array
HW	Hardware
PKI	Public Key Infrastructure
PUFs	Physical Unclonable Functions
PW	Password
QKD	Quantum Key Distribution
ReRAM	Resistive Random-Access Memory
RISC	Reduced Instruction Set Computer
RSA	Rivest–Shamir–Adleman encryption
SHA	Standard Hash Algorithm
SW	Software
TAPKI	Ternary Addressable Public Key Infrastructure
T-TRN	Ternary True Random Numbers