Exploiting Efficiencies in IoT Key Exchanges Through Reversible Logic Blockchains

John M. Medellin, PhD, CPA
Medellin Applied Research Concepts, LLC
Highland Village, Texas 75077, USA
john@mede-arc.com

Springer Book URL https://link.springer.com/book/10.1007/978-3-031-92625-9?sap-outbound-id=5D32C5C8DF19897E85A85EC97E9C3C4432890AC8&utm_source=standard&utm_medium=email&utm_campaign=000_LAN36_0000019083_Book%20author%20congrats%20NEW&utm_content=EN_34155_20250721&mkt-key=42010A0D55441FD098D077B33AEE40E2

Abstract. This article discusses the computational resource efficiencies gained in binary blockchain identity validation. Most blockchain algorithms using nonce mining are computationally intensive and continue increasing in resource usage as the chains get larger and more convolutional. This paper contains algorithms based on digital logic and parallel gated computation which are faster and lighter in resource consumption. A proposed theorem for reversible gate usage is presented and deployed to speed up key exchanges in blockchain architectures.

New experiment results with digital computers and gated arrays support the approach of distributing computation in binary secrets through blockchain blocks later to be exploited in reversible gates. The gated array's complete parallelization gives significant speed advantages over digital computer signal processing where interrupts must be used. In addition, blockchain algorithms must be translated from decimal representation to binary before computer math happens. Furthermore, digital chips will process instructions in stages versus parallel gates in gated arrays. The combination of these differences has significant impacts in performance and resource usage.

The FPGA (Field Programmable Gated Array) architecture is a physical example of a gated array architecture and is deployed in this continuing research. More resources are required for traditional key exchange versus reversible gates using blockchain to store previous binary keys. A previously published binary trust algorithm is deployed and scaled in both devices and compared to Diffie-Hellman/SHA256 key exchanges. The results point to usage of lighter binary protocols for identity validation when computational resources need to be conserved. This is of key importance in resource/power constrained environments and may be useful to consider in IoT and "Green" computing.

Keywords: Blockchain, FPGAs, Identity Validation, Computational Resources, IoT, Green Computing.

1. Introduction

Blockchain architectures and crypto-currencies have been accepted into the vernacular of technology and society, it seems that most people will know about these very soon if they don't already. The number of blockchain algorithms and associated uses for them has greatly increased their velocity in the past few years [1]. Most of these newer architectures take advantage of the greater resources available in processors by increasing complexity of their operations. As blockchains become longer and more complicated, they tend to consume more resources.

In response, algorithms which drive complexity to a lower level have been brought forth. In most cases, these algorithms advocate that what is required for the blockchain to process should be directly related to the identity and validation that might be required by the terms of service. In some cases, knowing that entity has valid identity is sufficient (for example in cases of protecting industrial machines) to validate the message that it transmits. These lighter "protocols" advocate using blockchain-type algorithms to drive trust and identity rather than confirm financial transactions such as those put forth in the crypto-currency world. Some examples of those are in [2] and [3]. This document makes uses of a light weight algorithm to determine if such an architecture could be better served in a gated array or digital computer type of processing environment.

This document is continuation of research provided in *https://www.athensjournals.gr/sciences/2022-9-1-2-Medellin.pdf* ; that document provides important background to the research presented here. This paper is organized into related work which includes: number systems and digital logic, identity, and blockchain as a means for establishing identity. Next, the discussion focuses on computation methods and architectures differentiating between digital computers and gated arrays. Finally, a simulation-experiment gives an example of the order of magnitude resource usage between the digital computer and gated array architecture in binary identity operation.

2. Related Work

Previous work is in three categories: numeric systems and digital logic, identity, blockchain as a means for establishing identity, and the data implications associated with storing blockchain blocks in memory.

2.1 Numeric Systems and Digital Logic

Human counting and machine counting differ significantly in their methods but not their answers. Where a human will typically count in bases of ten (1-10, 11-20 and so forth), a digital computer or other type of machine might count in binary (0 or 1). Even though both methods will lead to the same answer, each is implemented differently and these implementation approaches have significant differences.

Number systems can be discussed through their numeric bases: binary is in 0's and 1's, octal will count from 0 to 7, decimal from 0 to 9 and hexadecimal from 0 to F. Each system will place one of those characters in the representation beginning with the least significant on the right to the most significant being on the left. For example, the decimal number 16 is the power of 1 multiplied by 10 (2^{nd} digit to the left, the number 10) plus the remainder of 6 which is a total of 16. The same number in binary would be 10000 which is the number 1 times the power of 2^4 (2*2*2*2=16) plus zero. Both number systems give the same result but use different bases (2 and 10) to represent their count. Humans have been counting in decimal throughout history while machines count in binary because they count the high or low polarity of a particular cell in their memory or circuit representation [4].

The concept of digital logic is also important in machine operations. In basic form, the system consists of deriving results from interaction of two inputs that have either positive or negative polarity (ie: 1 or 0) through a logic gate. For example, if inputs are set to 1,1 and an OR operation (meaning "either or" in colloquial terms) it is interpreted as 1 OR 1, since both choices are a "1" then the outcome is a "1" from that operation and the same outcome is given when the inputs are (1 0) and (0 1). However, when the inputs are (0 0) then the only choice that can be selected is a

"0". Other gates exist, for example the AND gate specifies that both choices must be the same ex. 1 and 1 have the same outcome in that case being "1". Inputs (1 0), (0 1) and strangely enough (0 0) all yield "0", this final outcome being because colloquially "0" AND "0" means the only choice is a "0" between the inputs, the other two however correspond to negation (that they are not the same; 0 is not 1 and 1 is not 0). Commonly used gates are given in Fig.1[5].

	Inputs		Logic Gate Outputs (Z)					
Scenario	X	Y	OR	NOR	AND	NAND	XOR	XNOR
1	1	1	1	0	1	0	0	1
2	1	0	1	0	0	1	1	0
3	0	1	1	0	0	1	1	0
4	0	0	0	1	0	1	0	1

Fig. 1. *Inputs & Outcomes for Commonly Used Gates*

Sequencing together many input/gate/output combinations will result in patterns that are common to most mathematical operations. This science of sequencing these operations into streams that will produce certain desired outcomes is the principal objective of digital logic programming in modern computer systems. In that scenario, human language-like instructions are interpreted by a variety of tools to yield deconstructed operations that can be executed by machines to produce the desired outcome from a given operation[6]. Computer programming can be thought of instructing tools to decomposed desired outcome statements and execute them in logic gates[7].

2.2 Identity

Validation that a message comes from some entity that is in fact who they say they are is a critical property to achieve in secure environments. Several standards recommend that such validation of identity take place through security controls. Organizations such as NERC CIP (Nuclear Energy Commission/Critical Infrastructure Permanent standard) and ISO/IEC (International Standards Organization/International Electrotechnical Commission) standard 27002 require authentication to provide access. In the case of NERC CIP prescribes heavy penalties for non-compliance [Ind Network Security Book]. The identity property will be used to evaluate the two architectures proposed in the experiment.

Verification implies mathematical validation to a degree that the arithmetic operation performed can only result in one and only one outcome. This is a key proposition of modern cryptography [engineering cryptography]. Stallings [8] defines identity as a property of modulo operations as follows:

$$(0 + w) \bmod n = w \bmod n \quad (1)$$

$$(1 * w) \bmod n = w \bmod n \quad (2)$$

The above can be proven in binary numbers by: $0 \vee 1 \bmod 0 = 1$ ($1 \bmod 0 = 1$), $1 \wedge 1 \bmod 0 = 1$ ($1 \bmod 0 = 1$) or; $0 \vee 1 \bmod 1 = 0$ ($1 \bmod 1 = 0$), $1 \wedge 1 \bmod 1 = 0$ ($1 \bmod 1 = 0$). Coincidentally, using the XOR gate ($\oplus$) holds the same properties and will be discussed later in section 3.

2.3 Blockchain as a means for establishing Identity

Blockchain is a technology architecture pattern that enables sequencing and validation of events via computation. The concept was made famous by Nakamoto (an unknown entity) [9] but has roots in the byzantine generals problem. In that case, generals are confirming the time of attack on an adversary without having instructions from a higher officer. They establish communication

between themselves and at some point in time, there is sufficient consensus of when to attack [10]. This consensus process has been the subject of much study but in the Nakamoto paper it is derived when a particular entity (a "miner" in most cases) can provide the singly used number (a "nonce" [11]) to validate the specific encrypted block. In [9], the miner is an entity that provides the "Proof of Work", proof that it has done the work to find the encryption key that contains the nonce for the block being verified. Extrapolated to the byzantine generals problem, this would mean that there is a message to attack that has been mathematically verified by an entity that has done the work to verify and is therefore valid. A graphic example of blockchain blocks is given in Fig. 2.

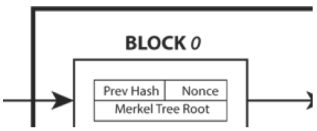

Fig. 2. *A Sample Partial Blockchain Sequence and Block*

Several methods of consensus have been proposed in addition to PoW, some are proofs of an activity (Proof of Computation "PoC" where the highest proportion of the computation resources in the blockchain network has been spent by one entity and therefore constitutes the ability to verify, e.g. "I did most of the work" [12]). These methods also tend to be computationally intensive, increasing in resource requirements as the blockchains get longer in number of blocks since most of the time either there are more participants (as in PoC above) or more blocks are added to the algorithm (PoW above; since there is a previous carry-forward element to each block).

Other methods of consensus have been proposed whereby the computational aspects might not be as severe as the examples cited above. Several of those methods constitute in appointment of one "master" or representative within the network who will validate the next block or set of blocks. These are sometimes appointed by receiving the most number of votes [13] or by random number assignment [14]. This article uses the second method for selection of a "machine-manager"; the appointment of one machine in the network which executes the tasks outlined below during it's time-dimensioned reign (or "epoch") as proposed in [16]. The tasks that are performed by the manager are:

1. Conduct the randomized election of the next machine-manager/backup manager.
2. Notify the two machines in #1 (obtain acknowledgement from them).
3. Notify the network of the election through a special key and operation
4. Issue the epoch's public key.
5. Issue private keys to newly admitted members.
6. Deprecate private keys to newly expired members.

7. Issue operations key (optional, based on storage requirements).
8. Hash closure of the epoch block.

The above are stored in the epoch's block and the epoch is considered closed until another one begins and machines communicate using the announced keys in their dialogue in Fig. 3.

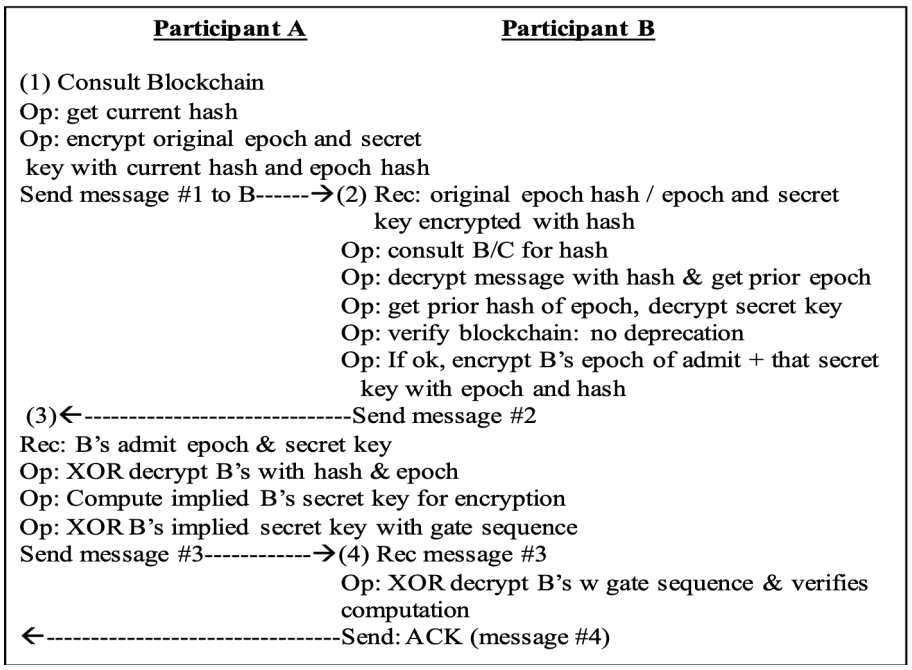

Fig. 3. *Handshaking Dialogue*

Guide: Op : Operation, Send : Transmit, Rec : Receive

The above method stores the key components for a time-dimensioned epoch in non-volatile memory for later retrieval and is significantly different than other methods such as Wei et. al. [15]. In that specific document, binary exchange keys are performed in a static computation method similar to the ones that will be tested in the experiment below (DH/SHA256). Furthermore, those methods do not use the concept of storing state from a previous dimensioned epoch for later use and retrieval in dialogue to validate identity. Identity is thus validated with a time dimension rather than specific primary key style exchange.

2.4 Blockchain Blocks and Storage Requirements

In addition to processing specific instructions via gates, the gated array will also necessitate storage for the blocks of the blockchain. The blockchain storage requirements for the model [17] and the sample block organization / traversals are in Fig.4.:

Component	Contents
Epoch ID	Sequential number for the epoch
Manager secret key and epoch	4096 bit key + original epoch of admission
Public key	4096 bit key generated by manager for the epoch
Gate sequence	4096 x 2 bit key corresponding to the gate being used
Admitted Secret Keys	Sequences of 4096 bits for new members
Deprecated epoch/keys	Deprecated sequences of previous members
Current hash	Previous hash XOR public key XOR gate sequence XOR manager epoch XOR manager secret key XOR current epoch XOR nonce

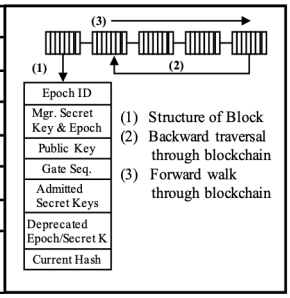

Fig.4. *Blockchain Block Components and Sample Traversals Through Memory*

The blockchain block used in estimating storage requirements in Fig. 4 are 6,832 bits per block (assumes no gate sequence and deprecated sequence).

3. Computational Methods and Architectures

This section is focused on discussing how logic and mathematics impact the computational load and how it is present in computer architectures. First, a discussion on human and machine operations presents the basic differences between how a human might arrive at an answer versus a machine. Next, a focus on digital logic leading to "reversible" gates and combinations that will help to explain how the two approaches; digital computer versus gated array might have some differences on the amount of resources used to derive a particular answer. This part of the article conveys the basic premises for the experiment that is presented in the subsequent section.

Validating identity in this document is defined as the ability to compute results on combinations of large chains of binary digits. The third focus of this section is to provide a discussion of the computational gates that are used in binary exchanges between gated arrays enabling them to validate the identity of an external communicating machine. Following the aspects of human vs machine operations, a specific discussion focuses on the properties of XOR and XNOR gates and the ability to create a model where if three of the components in the gate equation are known, the fourth will be able to be derived. Next, a discussion on the x86 instruction set execution is provided to set a common understanding of the 5 stage sequence processor architecture that is used by most common digital computers and the overhead associated. The final part of this section introduces two key computational aspects of gated array processing, the single stage instruction computation and the parallelism aspect of this architecture.

3.1 Human vs Machine Operations

Human beings perform arithmetic operations in a similar but uniquely different than digital or analog machines. Most human arithmetic is based on the decimal system (see above discussion), we count in 10's, add and subtract in 10's and express our thoughts in series of 10's. Computing machines however, count and perform operations on binary numbers (0/1) via logic gate comparators in order to derive and report results. A key part of both tool sets is the ability to shift characters (0/1 or 0-9) to the left in order to utilize the multiplicative "power of". As discussed above, a "1" in the left most of two digits signifies the number 10 plus whatever is on the right side of a two digit number (0 to 9; means 10 to 19 in human).

Machine operations also use basic symbols of addition/subtraction ("+" in human and OR in machine, NOR or OR NOT for "–" in machine) and multiplication/division ("X" in human and

AND or NAND for "/" in machine). In a very simplified example, the binary number 6 (0110) OR; added to the number 2 (0010) would go through the process of (from leftmost position):

1. 0 OR 0 is 0.
2. 1 OR 1 is 1 with a carry of 1 (0 for that digit because of the carry of 1).
3. 1 OR 1 (because of a carry of 1) is 1; but with a carry of 1, represent as 0.
4. 0 OR 0 is 0 but because of the carry of 1 is now a 1.
5. The final result is three 0's and a 1 in the left most bit (1000) and 8 in human count.

A similar process would be followed for multiplication except the gate to be used would be the AND gate. The same addition process can be achieved by "shifting" bits as follows (OR gate):

First (Input):	Second (Shift):	Third (Operate):
0110 = 6 → Original	1**1**00 ← Shift left 1 digit	1000 = 8 ←**1** s cancel out
0010 = 2 → Addition	0**1**00 ← Shift left 1 digit	

Then, substitute a 0 for the 3rd digit where both are the same (0 or 1) gives the result of 1000. This shifting operation is significantly less power consuming than performing the operation as done in the first part of the example since we are just shifting the position of the registers [18] this is ultimately how almost all processors execute arithmetic operations of addition. The power of using shifting and other logic encoded operations in their native state (without translation from human representation) is a resource saving tool that will be used in the experiment later in this document.

3.2 Associative Properties of XOR and XNOR Gates

The associative property of mathematics states that for either addition or multiplication equations [19]:

$$\{ (A \vee B) \vee C \equiv A \vee (B \vee C) \} \;\&\&\; \{ (A \wedge B) \wedge C \equiv A \wedge (B \wedge C) \} \tag{3}$$

therefore, by implication these operations:

$$\{ (A \oplus B) \oplus C \equiv A \oplus (B \oplus C) \} \;\&\&\; \{ (A \odot B) \odot C \equiv A \odot (B \odot C) \} \tag{4}$$

(the above are also valid without parentheses placement)

In the case of the XOR and XNOR gates, only 4 outcomes are possible with 3 input variables in either XOR or XNOR (in Table 1 below, "C" column corresponds to "0" for XOR and "1" for XNOR). The objective of the table is to first try to predict the value of a variable based on the other three variables provided. For example, in the column "Predict C" the value of C is being predicted in the second row based on the values of A,B and D (1,0,1) which yields the value of (0) for C according to the truth table. In the cell adjacent to the right, the standard value of (0,1,1=0) has been assigned, there are only 4 standard values in the table and all entries comply with that format. This has been done exhaustively throughout the table in order to determine if there are any other solutions than the four presented in the bottom row. Please note that only four outcomes exist when three binary inputs are provided.

Table 1. Standardized XOR and XNOR Operation Outcomes with Three Input Binary Digits

Input 1 [A]	Input 2 [B]	Operation [C]	Output [D]	Stdized. A,B,C=D	Predict C A,B,D=C	Stdized. Comb.	Predict A B,C,D=A	Stdized. Comb.	Predict B A,C,D=B	Stdized. Comb.
0	0	0	0	0,0,0=0	0,0,0=0	0,0,0=0	0,0,0=0	0,0,0=0	0,0,0=0	0,0,0=0
1	0	0	1	0,0,1=1	*1,0,1=0*	0,1,1=0	0,0,1=1	0,0,1=1	1,0,1=0	0,1,1=0
0	1	0	1	0,0,1=1	0,1,1=0	0,1,1=0	1,0,1=0	0,1,1=0	0,0,1=1	0,0,1=1
1	1	0	0	0,1,1=0	1,1,0=0	0,1,1=0	1,0,0=1	0,0,1=1	1,0,0=1	0,0,1=1
0	0	1	1	0,0,1=1	0,0,1=1	0,0,1=1	0,1,1=0	0,1,1=0	0,1,1=0	0,1,1=0
1	0	1	0	0,1,1=0	1,0,0=1	0,0,1=1	0,1,0=1	0,0,1=1	1,1,0=0	0,1,1=0
0	1	1	0	0,1,1=0	0,1,0=1	0,0,1=1	1,1,0=0	0,1,1=0	0,1,0=1	0,0,1=1
1	1	1	1	1,1,1=1	1,1,1=1	1,1,1=1	1,1,1=1	1,1,1=1	1,1,1=1	1,1,1=1
Standardized Binary Combinations:						**0,0,0=0**		**0,0,1=1**		**0,1,1=0**		**1,1,1=1**

In the case of the underlined & *italics* table cell by associative property:

Predictive Equation $\Rightarrow$ *Standardized Combination*

$$\{ 1 \oplus 0 \oplus 1 \} = 0 = \{ 0 \oplus 1 \oplus 1 \} \quad (5)$$

(from above, it is valid in any form of parentheses placement)

This is also the case for the rest of the table entries. Given the above the following theorem and proof are proposed:

Proposed Theorem: *"For any combination of three binary digits, 1 or 0 when the XOR or the NOR gate is used to evaluate the logic operation, the fourth digit shall be equal to 0 when any combination of the first three has all the values of 0 or only one of the first digits has the value of 0 else the value of the fourth digit shall be equal to 1".*

Proposed Proof:

$$\{ 0 \oplus 0 \oplus 0 \} = 0 \;\&\&\; \{ 0 \odot 0 \odot 0 \} = 0 \quad (6)$$

$$\{ 0 \oplus 1 \oplus 1 \} = 0 \;\&\&\; \{ 0 \odot 1 \odot 1 \} = 0 \quad (7)$$

$$\{ 0 \oplus 0 \oplus 1 \} = 1 \;\&\&\; \{ 0 \odot 0 \odot 1 \} = 1 \quad (8)$$

$$\{ 1 \oplus 1 \oplus 1 \} = 1 \;\&\&\; \{ 1 \odot 1 \odot 1 \} = 1 \quad (9)$$

The proposed theorem's reduction drives significant additional efficiency in computation. It also increases in importance when large chains of the gated operations serve to establish modular arithmetic identity (see section 2.2). The impact on processing of the length of the binary chains (used as keys in the experiment) can be much mitigated by deploying the concept described above; it is reduced to a simple "if" statement.

3.3 x86 Staged Architecture

As mentioned above, the x86 processor architecture is featured in modern digital computers. It forms the basis for ARM and Intel commercial processors which tend to be the most used in anything from laptop to tablets and other such devices. The architecture processes one instruction at a time in five stages [20] (Fetch (F), Decode (D1), Memory (D2), Execute (EX) and Writeback (WB)) shown in Fig. 5 below:

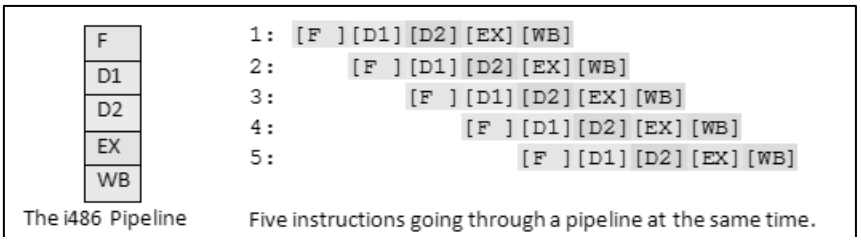

Fig. 5. *Five Stage Instruction Architecture (GameDev.net)*

This process is repeated throughout the life of a program, one instruction at a time. A detailed discussion of the above has been the topic of books and is beyond the scope of this document. The principal items to take forward from this sub-section are:

a. The instructions are executed one at a time, in stages with 3 to 5 CPU cycles.
b. The memory penalty can be high if the data is not in high level cache or registers.
c. The CPU operates faster than the gated array but not more efficiently.
d. The additional CPU resources to execute and these resources may be significant.
e. Additional cores (4-8 at the time of this writing) enable a level of parallel execution.

3.4 Gated Array Architecture

Gated arrays in contrast with digital computers contain only I/O Cells where data or instructions can be input) and logic blocks (gates for processing logical operations), some may be special gates for aggregation of results. Gated arrays are usually complemented with volatile memory (RAM) and sometimes with non-volatile memory (Solid State Disk Arrays, SSD) [21] Fig. 6 below [22] illustrates the basic architecture of the gated array.

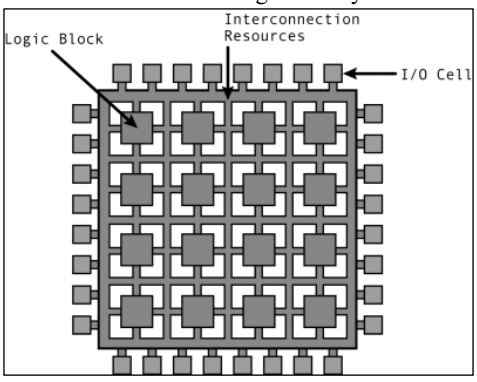

Fig. 6. *FPGA Conceptual Architecture*

Using Fig. 6, instructions for channeling to a XOR or XNOR logic block and necessary data for the operation would initiate the process through I/O cells. Next, the logic block would perform the operation between the two data points and channel the output either to another logic block (for further chaining of operations, for example an AB XOR→C XOR→D) or to an I/O cell. The instructions for directing this processing are created using HDL (Hardware Definition Language) and specifically indicate to the circuitry the sequence and usage of the gates to effect a desired condition. Gated arrays can range from thousands of gates (the one used in the experiment) up to millions for specialized (and higher price) models [23]. Gated arrays will typically operate at a

much slower speed (Megahertz vs Gigahertz) and also consume much less energy since they do not have the memory capacity or potentially the additional peripherals that other devices provide. From a throughput processing perspective, the following characteristics distinguish these devices:

1. Processing multiple banks of data in parallel without impacting indirection.
2. Processing concurrent vectors of data (many values in an array, all at once).

For our purposes, the above enable the processing of many XOR/XNOR values at once without incurring the 5 stage decomposition of each instruction. The parallel execution, concurrent operations, and native execution allows these processors to be able to compete and win in certain system architectures.

4. Proposed Experimental Models

The proposed blockchain experimental model is one where xor and xnor operations are used to derive results. As discussed in section 3.1 above, these particular gates can be de-aggregated into four specific terms, two inputs, one operation and one output, knowledge of 3 of them will provide the value for the fourth one with logical certainty (a tautology [24]). The specific model [25] proposes to store key values in 256 bits for each of the four values. One value is given as the private key (the first input for example) for the specific machine and stored in non-volatile memory at the local level. A second value (the second input for example) is the "epoch" value of same length and is generated each epoch by the machine-manager of that epoch (an epoch is a time-delimited segment where that key is valid). A third value constitutes of the operation value and may be issued by the manager from time to time instead of the epoch value. A fourth value (the output for example) may also be issued by the epoch manager-machine. The manager-machine may issue one or two values at the beginning of the epoch with the directive to substitute (in case of 2) or regenerate the epoch key (if only one is issued) internally based on the other two public values stored in the machine's memory.

In this specific document, the research is focused on fetching blocks from memory and processing them through XOR or XNOR gates directed by the gate sequence chain only. These are the major components of the dialogue in Figure 3.

4.1 Comparison Models

The Diffie-Hellman and the SHA256 variation will be used to compare vs the one described in this document. Both those algorithms use discrete modular mathematics to arrive at a particular key which will be used for authentication and cryptography of messages. Traditional Diffie-Hellman will be used in the FPGA experiment and the SHA256 will be used for the digital computer version. An example of Diffie-Helman is as follows [26]:

a. Alice and Bob share a prime number (353) & a 2nd number less than the first (3).
b. Alice and Bob generate a product that is less than the prime (97 and 233).
c. They calculate the second ^ third Modulo the prime number and exchange that product (248 and 40).

d. The exponentiate the number generated in c to the power of the number received in c Modulo the prime number (160 for both) and that is their shared key for encryption and authentication.

The above operation generates 2 operations in (a), 1 in (b), 1 in (c) and 1 in (d) for a total of 5 operations required from each. The SHA256 computation is similar to the traditional Diffie-Hellman and more readily available in in modern software, that will be used in the digital computation. In contrast, the key generation algorithm (including the RAM memory fetch) will be scaled 4 times per interchange for comparable operation (see sections above).

4.2 Simulation Environments

Two simulation environments were provided to test the algorithms discussed in this section:

1. Apple Macintosh MacBook Pro 2021, Apple M1 Pro Chip (similar to Intel corei7), 16GB of RAM.
2. Xilinx Baysys 32 FPGA board, 32Mb RAM, 280,000 logic gates.

The monitoring and Interactive Development Environments (IDE) were as follows:

1. Linux “top” command, Visual Studio Code 1.87, Source: Golang, Compiled: C.
2. Vivado Xilinx CAD FPGA, through synthesis. DH configuration in Verilog HDL, this document’s configuration in VHDL (both simulated and synthesized to gate configurations in the vivado toolkit).

Both the above have monitoring tools for determining resource usage.
The code models that are being used are:

1. GOLANG Diffie-Hellman: “GOLANG CRYPTO/ECDH AND THE TPM” [27].
2. Verilog Diffie-Hellman: “borabarduk/KeyExchangeFPGA” [28].
3. GOLANG Binary Key Exchange: Appendix A.
4. VHDL Binary Key Exchange: Appendix B.

5. Experiment Results

The models were executed in their respective environments with the following results.

5.1 Digital Computer Results

The two GOLANG models were compiled into C executable binaries and their execution results and sources are documented in Table 2. The keys generated by the two algorithms are 1024 bits (64 Hexadecimal Characters).

Table 2. Digital Computer Results (D.H.=Diffie-Hellman, Gate=This Document)

Number of Exchanges	D.H. CPU Time	Gate CPU Time	D.H. Memory	Gate Memory	D.H. Threads	Gate Threads
1000	0.13 sec	0.02 sec	3457K	2993K	5	5
5000	0.56 sec	0.05 sec	6833K	3009K	6	5
10000	1.10 sec	0.09 sec	7329K	2977K	7	5
15000	1.63 sec	0.11 sec	7745K	3041K	7	5
20000	2.16 sec	0.14 sec	7649K	2849K	7	5
30000	3.23 sec	0.19 sec	7793K	2849K	7	5
40000	4.30 sec	0.24 sec	8609K	2849K	7	5
50000	5.36 sec	0.30 sec	7777K	2945K	7	5
100000	10.70 sec	0.58 sec	8225K	2945K	7	5

5.2 Field Programmable Gated Array (FPGA) Results

The two hardware definition language models were simulated for correctness and synthesized for hardware specific configuration Xilinx xc7a35ti. Their execution results and sources are documented in Fig. 7 and Table 3.

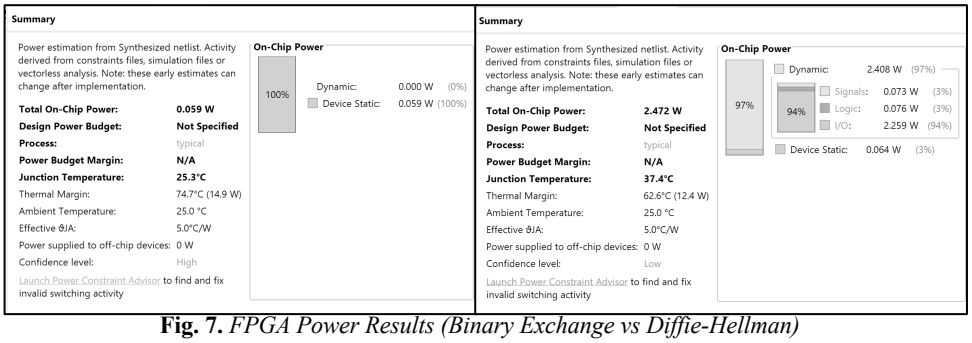

Fig. 7. *FPGA Power Results (Binary Exchange vs Diffie-Hellman)*

The screen shots in Fig. 7 are from the binary key generation (left) and the Diffie-Hellman keys generation. In the binary key we see the usage of power is largely on the device static front since the processing of the binary exchange is done almost instantaneously in one operation. The one on the right displays the mathematical overhead of generation and exchange of keys through the DH algorithm (most of the power is dynamic and used in application logic).

Gated Array	Power Used	CPU Time	Memory
D.H.	2.472 W	0.27 sec	1470K
Gate	0.059 W	0.05 sec	1475K

Table 3. FPGA Results One Key Exchange

Table 3 summarizes the results of the FPGA experiment indicating that there is a significant power savings in using the binary gated algorithms vs generation of traditional DH keys. A further discussion is provided in section 5.3 below.

5.3 Results Discussion

The four experiments above strongly point to significant efficiencies in using the gated key exchange versus the Diffie-Hellman algorithm. The probability of attack success using brute force style for that algorithm is given in [17] and was reported at a range of 2^16 to 2^1,048,576 depending on number of valid keys in an epoch and the time interval between epochs and regeneration of keys. Continuous regeneration of keys for the gated algorithm would definitely cause more resources to be consumed and that specific aspect of the model was not tested in this experiment. The scope of testing was given to key generation and exchange between two machines with different algorithms and in that case the gated algorithm has a very clear advantage. Caution is given to future implementations to test further with specific service level agreements and probability of attack success vs resource usage (emphasis added).

In the case of the digital computer, the operating system (os) increments the number of threads very quickly from 5 to 7 (the maximum available for application execution is 7) and increases in memory are seen through the rest of the experiment. This is due to increased memory requirements in temporary storage due to saturation of the computation space. Key generation and exchange remains fairly stable at a rate of 0.13 / 1000 to 0.11 / 1000 (the last iteration: 10.71/100,000). That compares to the os only assigning 5 threads to the gated algorithm and stabilizing at around 3,000K in memory and consistent generation of keys at the rate of 0.02 / 1000 to 0.006 / 1000 (the last iteration again: 0.58 / 100,000). Given these statistics, the gated execution has dramatic advantages over the Diffie-Hellman in generation and exchange of keys. These advantages would also translate into power savings (however, power savings is not as important in digital computers because they can be linked to an electrical outlet fairly easily).

In the case of the FPGA, the results are also quite dramatic in execution times and consumption of power. Because of the parallel nature of FPGAs the memory aspect does not need management in a similar fashion as when threads are saturated and more swapping of memory needs to happen (digital computers). Because of the key size, the memory requirements are fairly similar (almost the same) in both executions. However, the low computational aspects of the gated algorithm translate into significant time of execution and power requirements. These are key attributes in the Internet of Things (IoT) and also in Green Computing where conservation of energy whilst maintaining SLA is paramount to operation of the system.

6. Future Work

This document focused on one part of the execution aspects of a security system. Both algorithms are self-contained when it comes to validation of identity. The study did not delve into the aspects of key management or regeneration which also require significant resources (especially in the case of the gated algorithm where the parameters are only valid in a particular epoch). A key characteristic of the gated algorithm is the appointment of managers who generate/regenerate keys for future epochs. The manager elections, the manager tasks and the communications between the IoT devices will also need to be modeled and that is an aspect of the future work that has to be addressed.

The simplicity of the algorithm is obfuscated by this randomness both in key size, epoch duration, manager election, addition/deprecation of members and other aspects that together will have a strong impact on performance and attack resiliency. These aspects would need to be also modeled before any field implementation is to be undertaken.

7. References

1. J. Medellin, M. Thornton: "Performance Characteristics of Two Blockchain Consensus Algorithms in a VMWare Hypervisor" 2018 International Conference on Grid & Cloud Computing and Applications "GCA '18", p. 10 - 17.
2. T. Salman, M. Zolanvari, A. Erbad, R. Jain, M. Samaka: "Security Services Using Blockchains: A State of the Art Survey" IEEE Communications Surveys & Tutorials 21(1), p. 858 – 880.
3. A. Dorri, S. S. Kanhere, R. Jurdak, P. Gauravaram: "Blockchain for IoT security and privacy: The case study of a smart home" 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), p. 618 – 623.
4. Readler, Blaine C.: "VHDL by Example; A Concise Introduction for FPGA Design" Full Arc Press, Yorkshire, UK (2014).
5. Electronic Projects Focus, https://www.elprocus.com/basic-logic-gates-with-truth-tables , last accessed on 16 February, 2021.
6. Chu, P.: "FPGA Prototyping by VHDL Examples" Wiley Publishing, Inc., Hoboken, New Jersey (2008).
7. Haskell, Richard E., Hanna, Darrin M.: "Digital Design; Using Digilent FPGA Boards – VHDL/Vivado Edition. 4th ed." LBE Books, Auburn Hills, MI USA (2018).
8. W. Stallings: "Cryptography and Network Security, Principles and Practice 7th edition" Pearson Education Limited, London, United Kingdom (2018).
9. "Bitcoin: A Peer-to-Peer Electronic Cash System", http://www.bitcoin.org, last accessed March 19, 2019.
10. L. Lamport, R. Shostak, M. Pease: "The Byzantine Generals Problem" ACM Transactions on Programming Languages and Systems 4(3), p. 382-401.
11. N. Ferguson, B. Schneier, T. Kohno: "Cryptography Engineering, Design Principles and Practical Applications" Wiley Publishing, Inc., Indianapolis, Indiana (2010).
12. K. Yeow et. al." Decentralized Consensus for Edge-Centric Internet of Things: A Review, Taxonomy, and Research Issues" IEEE Special Section On Internet-Of-Things (Iot) Big Data Trust Management, Digital Object Identifier 10.1109/ACCESS.2017.2779263 (2017).
13. D. Ongaro and J. Ousterhout: "In Search of an Understandable Consensus Algorithm" Proceedings ATC'14 USENIX Annual Technical Conference (2014), USENIX, p. 305-319.
14. J. Medellin, M. Thornton: "Simulating Resource Consumption in Three Blockchain Consensus Algorithms" "MSV '17" International Conference on Modeling, Simulation & Visualization Methods, p. 21 – 27.
15. Wei et. a.: "FIELD-PROGRAMMABLE GATE ARRAY BASED TRUSTED EXECUTION ENVIRONMENT FOR USE IN A BLOCKCHAIN NETWORK" US Patent 10,657,293 B1 (May 19, 2020).
16. Medellin, John M. "Generation, Regeneration and Validation of Binary Secret Keys through Blockchain in IoT Devices" 17th Int. Conf. on Information Technology & Computer Science, ATINER, Greece. https://www.atiner.gr/2021educom-pro, last accessed April 19, 2024.

17. Medellin, John M. ""Generation, Regeneration and Validation of Binary Secret Keys through Blockchain in IoT Devices" Athens Journal of Sciences 9(1), 25-46 (2022).
18. Monk, S.: "Programming FPGAs, Getting Started with Verilog" McGgraw Hill, New York (2017).
19. Johnsonbaugh, R.: "Discrete Mathematics" 8th ed. , Pearson Education, Inc., New York (2018).
20. Hennessy, John L., Patterson, David A.: "Computer Architecture; A Quantitative Approach" 5th ed., Elsevier, Waltham, MA (2012).
21. ARM MPS3 FPGA Prototyping Board Technical Reference Manual, https://developer.arm.com/documentation/100765/0000/Hardware-description/User-non-volatile-memory last accessed April 19, 2024.
22. All About FPGAs, https://www.eetimes.com/all-about-fpgas/ last accessed 2024/04/10.
23. Serial NOR Flash (SPI) Memory, https://www.futureelectronics.com/c/semiconductors/memory--flash--norflash--serial/products last accessed 2024/04/19.
24. Meinel, Christoph, Theobald, Thorsten: "Algorithms and Data Structures in VLSI Design" Springer-Verlag, Berlin (1998).
25. Medellin, John: "DEVICE FOR IMPLEMENTING GATED ARRAY BLOCKCHAIN PROTECTION CODES FOR IOT DEVICES" US Patent Application 63215726, EFS ID: 43107077 (2021).
26. W. Diffie, M. Hellman "Multiuser Cryptographic Techniques" IEEE Transactions on Information Theory, November 1976.
27. https: //linderud.dev/blog/golang-crypto/ecdh-and-the-tpm/, last accessed 2024/05/02.
28. https: //github.com/borabarduk/KeyExchange , last accessed 2024/05/02.

About the Author

John M. Medellin received his Ph.D. with focus on Software Engineering and minor in Computer Engineering from Southern Methodist University. He is a retired Senior Executive at International Business Machines and also at PricewaterhouseCoopers. From 2015 to 2020 he was Research Associate Professor at Southern Methodist University (Darwin Deason Institute for Cyber Security) and Associate Professor at the University of Mary Hardin-Baylor. This is his ninth publication in the blockchain domain.

APPENDIX

A. Golang Code

```
package main
import (
"fmt"
"math"
"os"
"os/exec"
"strconv")
func main() {
keySize :=1 ; numIter := 1 ; productIs :=0 ; sum :=0 ; printMe :=0
```

```
fmt.Println("Format is key size, num conversions, 1 for print detail or 2 for print diagnostics only
(linux)")
if len(os.Args[1])!=0{keySize,_ = strconv.Atoi(os.Args[1])}
if len(os.Args[2])!=0{numIter,_ = strconv.Atoi(os.Args[2])}
if len(os.Args[3])!=0{printMe,_ = strconv.Atoi(os.Args[3])}
var x [16384]int ; var y [16384]int ; var z [16384]int
for i := 0 ; i < 16384 ; i++ { // generate 3 values in the arrays, fill them
x[i]= i % 2  ; y[i]= int(math.Abs(float64(x[i]-1 % 2)))
if int(math.Log(float64(i)))&1 == 0 {z[i]=1}
}
for j := 0 ; j < numIter ; j++ { // for number of key conversions to be done
if printMe==1{fmt.Println("Key Conversion Number",j+1)}
for i := (keySize*j%4) ; i < keySize + (keySize*j%4) ; i++ { // for the key size, generate key
// if x[i]+y[i]+z[i]%2==0 || x[i]+y[i]+z[i]==2{productIs=0}else{productIs=1}  // commented out:
computed using article procedure
if x[i]+y[i]+z[i]%3<2 {productIs=1}else{productIs=0} // computed using modulo operation per
article
if printMe==1{fmt.Println("x",x[i],"y",y[i],"z",z[i],"res",productIs)} // verbose output
}
sum ++
}
if printMe==1||printMe==2{fmt.Println("number of conversions",sum,"key size",keySize)} //
verbose output
if printMe==1||printMe==2{out,_ := exec.Command("top","-c","d","-F","-R","-o","cpu","-
l","1","-n","3").Output()
output := string(out[:])
fmt.Println(output)} // verbose output for linux
}
```

B. **VHDL Code**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity bin_gen1 is
end entity bin_gen1;

architecture Behavioral of bin_gen1 is
begin

 process is
-- Reg_Proc: process (clk)

variable in_1 : std_logic_vector (1023 downto 0);
variable in_2 : std_logic_vector (1023 downto 0);
```

```
variable out_1 : std_logic_vector (1023 downto 0);
variable i    : integer;
begin

in_1 :=
"00110011001100110011001100110011001100110011001100110011001100110
01100110011001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100110011001
10011001100110011001100110011001100110011001100110011001100110011
00110011001100110011001100110011001100110011001100110011001100110
01100110011001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100110011001
10011001100110011001100110011001100110011001100110011001100110011
00110011001100110011001100110011001100110011001100110011001100110
01100110011001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100110011001
10011001100110011001100110011001100110011001100110011001100110011
00110011001100110011001100110011001100110011001100110011001100110
01100110011001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100110011001
1001100110011001100110011001100110011001100110011";
in_2 :=
"01010101010101010101010101010101010101010101010101010101010101010
10101010101010101010101010101010101010101010101010101010101010101
01010101010101010101010101010101010101010101010101010101010101010
10101010101010101010101010101010101010101010101010101010101010101
01010101010101010101010101010101010101010101010101010101010101010
10101010101010101010101010101010101010101010101010101010101010101
01010101010101010101010101010101010101010101010101010101010101010
10101010101010101010101010101010101010101010101010101010101010101
01010101010101010101010101010101010101010101010101010101010101010
10101010101010101010101010101010101010101010101010101010101010101
01010101010101010101010101010101010101010101010101010101010101010
10101010101010101010101010101010101010101010101010101010101010101
01010101010101010101010101010101010101010101010101010101010101010
10101010101010101010101010101010101010101010101010101010101010101
01010101010101010101010101010101010101010101010101010101010101010
1010101010101010101010101010101010101010101010101";
out_1
:="000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000
```

```
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000";

 for i in 1 to 2 loop
 report "i-" & integer'image(i);
if i>0 then
  out_1 := in_1 xor in_2;
  end if;
  report "1-" & std_logic'image(in_1(1023-i));
  report "2-" & std_logic'image(in_2(1023-i));
  report "O-" & std_logic'image(out_1(1023-i));
 end loop;
wait;
end process;
end Behavioral;
```