Digital Bimodal Function: An Ultra-Low Energy Security Primitive

Teng Xu, James B. Wendt, and Miodrag Potkonjak
Computer Science Department
University of California, Los Angeles
{xuteng, jwendt, miodrag}@cs.ucla.edu

***Abstract*—We have developed a new security hardware primitive named digital bimodal function (DBF) that enables ultra low energy security protocols. DBF allows the computation of legitimate communicating sides to be compact and low-energy while it requires any attacker exponential computational effort and energy expense. Our new approach is competitive with the energy efficiency of traditional security key cryptographic security technique (e.g., AES) while more than three orders of magnitude more energy efficient than RSA. The implementation is demonstrated using the Xilinx FPGA platform.**

***Keywords*—DBF; FPGA; hardware security; public key cryptography;**

I. Introduction

Low energy and security have emerged as two paramount metrics for a variety of systems ranging from wireless sensor nodes and smart phones to laptops and data centers. Therefore, the importance of creating secure low energy platforms and protocols is widely acknowledged. There are a number of lightweight protocols for several secret key cryptographic primitives [1][2][3]. However, there are almost no energy efficient public key cryptographic primitives that are competitive with the energy efficiency of the traditional secret key primitives.

Process variation (PV) is an unavoidable side product of modern silicon implementation technologies. It is based on the fact that each transistor, gate, and wire on each integrated circuit with a particular design has unique physical (e.g. channel length) and manufacturing (e.g. leakage energy and delay) properties. A physically unclonable function (PUF) [4] is a deterministic multiple-input multiple-output system that is hard to reverse engineer and simulate. PUFs utilizing PV to create a hardware-based secret key primitive were proposed by [5][6] which are intrinsically resilient to physical and side channel attacks [7][8]. More recently, public PUFs [9] and SIMPL [10] create PUF-based security primitives and public key protocols [11][12][13] by employing huge gaps between execution and simulation. Many other PUF-based security primitives are also emerged and applied in areas of remote communication, sensor networks, and random number generator [14][15][16]. However, they require very large computation efforts that result in high energy requirements for at least one and often both participating parties [17].

Recently, the matched PPUF (mPPUF) [18] has been proposed as a new public key security primitives. mPPUF uses both process variation and device aging to create pairs of identical PPUFs that can be matched only with negligible small probability. It is a very energy efficient security primitive that can be used in a variety of cryptographic protocols. However, mPPUF poses high implementation requirements in terms of measurement accuracy and environmental stability.

Our technical goal is to take the next step in the development of low power security primitives and protocols by taking the best properties of each of the previously proposed solutions as our objectives. Specifically, in addition to our primary objective of creating solutions that require significantly lower energy, our goal includes public key security that does not require initial exchange of secret keys. The approach must be stable regardless of operational and environmental conditions as well as device aging. Hence, we pursue only completely digital techniques. In addition, we impose low energy requirements on all participating parties and look for flexible and small footprint solutions.

The core idea for our new approach is the notion of digital bimodal functions (DBFs). We define DBF as a mapping of randomly generated Boolean functions that has two forms: $f_{compact}$ and f_{expand}, among which $f_{compact}$ can be computed rapidly and requires only a small amount of energy while f_{expand} can only be computed using a very high amount energy, hardware resources, and an unacceptable amount of time. Due to the difference in computation complexity, we use $f_{compact}$ as the private key while f_{expand} as the public key in our public key protocols. A detailed example of DBF and a public key communication protocol is shown in the next section.

II. A Motivational Example

A. An Example of $f_{compact}$ and f_{expand}

Example I:

$$Input : a_i, i \in \{0, 1, 2...11\}; \tag{1a}$$

$$Output : c_j, j \in \{0, 1, 2, 3\}; \tag{1b}$$

$$\begin{cases} b_0 = a_0a_1a_2 + \bar{a_1}a_3 \\ b_1 = a_4\bar{a_5}a_6 + \bar{a_4}\bar{a_7} + a_5a_7 \\ b_2 = a_8\bar{a_9} + a_9\bar{a_{10}}\bar{a_{11}} \\ b_3 = \bar{a_0}\bar{a_4}a_8 + a_0\bar{a_{10}} \end{cases} \tag{1c}$$

$$\begin{cases} c_0 = b_0b_1b_2 + \bar{b_1}b_3 \\ c_1 = b_1\bar{b_0}b_3 + \bar{b_1}\bar{b_3} + b_0b_2 \\ c_2 = b_1\bar{b_3} + b_2\bar{b_0}\bar{b_1} \\ c_3 = \bar{b_0}\bar{b_2}b_3 + b_1\bar{b_3} \end{cases} \tag{1d}$$

$$\begin{cases} c_0 = a_0a_1a_2a_4\bar{a_5}a_6a_8\bar{a_9} + a_0a_1a_2a_5a_7a_8\bar{a_9} + a_0\bar{a_{10}}a_4\bar{a_5}\bar{a_6} \\ +a_0a_1\bar{a_{10}}\bar{a_{11}}a_2a_9 + a_0\bar{a_{10}}a_4a_5\bar{a_7} + a_0a_1a_2\bar{a_4}\bar{a_7}a_8\bar{a_9} \\ +a_0\bar{a_{10}}\bar{a_4}\bar{a_5}a_7 + \bar{a_0}\bar{a_4}\bar{a_5}a_7a_8 + \bar{a_1}a_3a_4\bar{a_5}a_6a_8\bar{a_9} \\ +\bar{a_1}a_3a_5a_7a_8\bar{a_9} + \bar{a_1}a_3\bar{a_4}\bar{a_7}a_8\bar{a_9} + \bar{a_1}\bar{a_{10}}a_{11}a_3a_4\bar{a_5}a_6a_9 \\ +\bar{a_1}\bar{a_{10}}\bar{a_{11}}a_3a_5a_7a_9 + \bar{a_1}\bar{a_{10}}a_{11}a_3\bar{a_4}\bar{a_7}a_9; \\ c_1 = \cdots; \\ c_2 = \cdots; \\ c_3 = \cdots; \end{cases} \tag{1e}$$

The new low energy security primitive is illustrated using Example I. There are four bimodal Boolean functions c_0, c_1, c_2, c_3 in (1d) created by using the variables b_0, b_1, b_2, b_3 in (1c). Each $b_j, j \in \{0, 1, 2, 3\}$ is created using a subset of inputs $a_i, i \in \{0, 1, ..., 11\}$. By substituting (1c) into (1d), we first expand each Boolean function of $c_j = f_j(a_0, ..., a_{11}), j \in \{0, 1, 2, 3\}$ to a sum of products and then simplify each function by using the tool Simple Solver to get the results shown in (1e). Note that the functions in (1e) can also be in a product of sums, we only show a sum of products here due to the space. The iterative substitutions and expansions between Boolean functions can create large functions that require many operations. In Example I, $f_{compact}$ requires 26 AND operations and 10 OR operations while in f_{expand}, 339 AND operations and 68 OR operations are required. We use the original functions before substitutions and expansions as $f_{compact}$ and the expanded functions as f_{expand}. In Example I, the functions in (1c) and (1d) are $f_{compact}$, while the functions in (1e) are f_{expand}.

Given the same inputs, both $f_{compact}$ and f_{expand} produce the same outputs, but the functions in (1c) and (1d) are much more compact in size while the functions in (1e) are large in size and can not be algebraically simplified. One further observation is that both (1c) and (1d) have the similar forms of functions that if we replace the inputs in (1c) with the inputs in (1d), (1c) can turn into (1d). Therefore, we only need one group of functions to represent the format of $f_{compact}$, e.g. (1c), which allows $f_{compact}$ to be even more "compact".

Example I offers us insight into how substitutions and expansions can create the large size difference between $f_{compact}$ and f_{expand}. However, to further show the impact of substitutions and expansions, we quantify the difference by first defining some criterion for comparisons.

One criterion is that we put both the $f_{compact}$ and the f_{expand} into the form of a sum of product and simplify them by using the tool Simple Solver, then we compare the total number of products. Note that the simplification procedures reduce the size of the Boolean functions, but only by a constant factor. In Example I, after simplification, $f_{compact}$ can be expressed by the group of functions in (1c), which has 9 products, and f_{expand} can be expressed by the functions in (1e), which have 72 products. We also define two potential dominate factors that influence the number of products of $f_{compact}$ and f_{expand}. One is the number of iterations, which equals to the level of substitutions between the groups of functions. The other is the number of primary inputs, in Example I, the number of primary inputs is 12 and the number of iterations is 1.

Based on the above definition, given the number of iterations and primary inputs, we randomly generate a group of functions as $f_{compact}$ and the corresponding f_{expand}, then compare the number of products. We make two restrictions to quantify the generation of $f_{compact}$ and f_{expand} in the test. The first is that for each single sub-function in $f_{compact}$, it can only involve at most 4 inputs, just as shown in Example I, which guarantees the size of $f_{compact}$ to be relatively small. The other restriction is that in each iteration, the number of inputs is the same as the number of outputs, e.g. we use a_i, b_i and c_i, $i \in \{0, 1...11\}$.

# of iterations	# of inputs	avg. # of products ($f_{compact}$)	avg. # of products (f_{expand})
4	8	23.2±2.8	259.2±16.1
5	10	28.7±3.1	765.6±68.7
6	12	34.6±3.5	3816.0±245.2
7	14	39.0±3.8	11454.8±758.1
8	16	46.7±4.1	49206.4±2684.8
9	18	52.1±4.3	142491.6±7520.1
10	20	57.8±4.6	369656.8±19265.5

TABLE I: Size comparison between $f_{compact}$ and f_{expand} with different number of iterations and different number of primary inputs. The average number of products are tested with 95% interval confidence.

The size difference between $f_{compact}$ and f_{expand} directly determines their difference in computation complexity. According to Table I, when the number of iterations is 10 and the number of primary inputs is 20, the average number of products in $f_{compact}$ is 57.8 while the average number of products in f_{expand} reaches 369656.8 which is 6395 times larger. Based on the results in this table, by increasing the number of iterations and the number of inputs, the number of products in $f_{compact}$ grows linearly while the number of products in f_{expand} grows exponentially, which offers us a very easy way to create a computation gap between $f_{compact}$ and f_{expand}.

B. Public Key Communication Using DBF

Public key communication is one of the most widely used communication protocols. We use it as an example to explain how DBF works as a security primitive in security protocols. The basic setting for this protocol is shown below.

- Private Key $-$ $f_{compact}$, denoted by K_{priv}.
- Public Key $-$ f_{expand}, denoted by K_{pub}.
- Alice $-$ the owner of K_{priv}. The party to receive and decrypt messages.
- Bob $-$ the party to send and encrypt messages.
- TTP $-$ trusted third party. The party that administrates K_{pub}.

Before explaining how this protocol works, we need to clarify one fact about $f_{compact}$ and f_{expand} which is that given $f_{compact}$, it is easy to calculate f_{expand} by making substitutions and expansions, but given f_{expand}, it is impossible to deduce $f_{compact}$ due to the difficulty of function decomposition. This is important because it guarantees the confidentiality of K_{priv}.

According to Protocol 1, the core idea of this protocol is to take advantage of the calculation time advantage of

Protocol 1 Public Key Communication

1: Bob has a message m to send to Alice, m is in the form of binary vector.
2: Suppose l single Boolean sub-functions are in K_{pub}, all of which have the form of a sum of products and a product of sums. In the first round, for the ith ($i \in\{1, 2, ..., l\}$) function f_i in K_{pub}, Bob randomly requests either one product from the a sum of products or one sum from the a product of sums from TTP. According to that product or sum, Bob generates a binary input vector p_i, making $r_i = f_i(p_i) = 1$ (case of product) or $r_i = f_i(p_i) = 0$ (case of sum). Bob connects p_1 to p_l to generate binary vector P_1 and connects r_1 to r_l to generate binary vector R_1.
3: Bob repeats step 2 for N rounds, generates P_j and R_j ($j \in\{1, 2, ..., N\}$).
4: Bob calculates $E = m \oplus R_1 \oplus R_2 ... \oplus R_N$.
5: Bob broadcasts E and all P_j ($j \in\{1, 2, ..., N\}$).
6: Alice computes all the sub vector p_i ($i \in\{1, 2, ..., l\}$) in each P_j ($j \in\{1, 2, ..., N\}$) with K_{priv} to find out the corresponding value of r_i, then Alice connects all the r_i to form each binary vector R_j.
7: Alice computes $m = E \oplus R_1 \oplus R_2 ... \oplus R_N$ and gets Bob's message.

$f_{compact}$ compared to f_{expand}, making the only person who can calculate R_j ($j \in\{1, 2, ..., N\}$) in reasonable time is someone who has K_{priv}. Another important design is that we use the trusted third party to administrate K_{pub}. Note that once Bob wants to send messages to Alice, he only needs to request one product or one sum of each function in K_{pub} from TTP, then creates the binary vector to encrypt his messages. This takes advantage of the property of the sum of products and the product of sums, which is that by knowing one product in the sum of products, one input vector that makes the output to be 1 can be deduced and by knowing one sum in the product of sums, one input vector that makes the output to be 0 can be deduced. This design minimizes the key size as well as the energy for calculation that Bob requires during the encryption of public key communication. However, for an attacker, if he/she wants to find out the right c_x in C, since he/she does not have K_{priv}, he/she can only request all the information of K_{pub} from TTP to simulate. Therefore, for an attacker, the public key size and the calculation scale are not minimized at all, the expected effort is that the attacker has to scan half of the a sum of products as well as half of the a product of sums, which is exponentially more efforts than Bob takes. Here the N rounds of operation is used to boost the calculation expense of $f_{compact}$ and f_{expand} N times simultaneously. In each round, corresponding P_j and R_j ($j \in\{1, 2, ..., N\}$) that are independent from any previous round are generated for encryption. As it takes much more time to calculate f_{expand} than $f_{compact}$, after both format increasing N times, the calculation time for $f_{compact}$ is still trivial but the calculation time for f_{expand} increases significantly. N is a flexible number that can be adjusted according to the size of $f_{compact}$ and f_{expand}.

III. Architecture on FPGAs

FPGA is a platform that provides a wide spectrum of advantages including low-power, low-cost, and self-trust. Moreover, the reconfigurability of FPGAs allows the system to be upgraded which is not possible with ASICs. We consider implementing $f_{compact}$ and f_{expand} on FPGAs respectively. The desiderata is to have low energy, delay, and area implementation of $f_{compact}$ while proving that there does not exist a fast and compact implementation of f_{expand}, therefore it can only be simulated.

Note that the configurable logic blocks (CLBs) which contain LUTs and flip-flops are the fundamental logic units in Xilinx FPGA. Specifically, for each LUT in the CLB, suppose it has 4 inputs, which is shown in Figure 1, its output can be expressed by a Boolean function f with the corresponding inputs. Now compare the function in Figure 1 with each sub-function in (1c), note that each sub-function in (1c) is a Boolean function with 4 inputs, which can be expressed exactly by the function format in Figure 1. Based on the above observation, we conclude that each function in (1c) as well as in (1d) can be implemented by a 4-input LUT. Therefore, we can use a group of 4-input LUTs to implement the functions in (1c) and (1d), whose architecture is shown in Figure 2.

The design consists of two levels of LUTs, each LUT implements one function in (1c) or (1d). A substitution is implemented by connecting the outputs of previous level LUTs to the inputs of the next level LUTs. To extend Example I to a more general case, any $f_{compact}$ that consists of iterations of groups of Boolean functions can be implemented by this interconnected LUT network on an FPGAs. In such combinational logic architecture, every iteration in $f_{compact}$ requires an additional level of LUTs.

An alternative design is to format the group of functions in each iteration the same, like (1c) and (1d), the only change is the inputs and outputs. In such cases, a sequential logic design is available. For example, the functions in (1c) and (1d) can be implemented by the architecture in Figure 3. The design uses only one level of LUTs and a group of flip flops are used to store the outputs of the LUTs. Each repeated cycle is equivalent to make substitutions between Boolean functions.

Compared with the combinational logic design, the sequential logic design has the advantage of more compact area and lower power. It is because the sequential design does not require additional area resources for iterations other than for the first round. Therefore, for n iterations, we only need to run the sequential architecture for n cycles. We introduce the notion of sequential logic clusters (SLCs) to denote this sequential architecture.

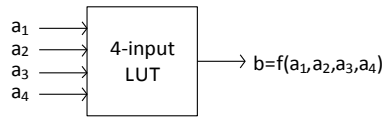


Fig. 1: The 4-input LUT: a_1, a_2, a_3, a_4 are the inputs, b is the output. The relation between b and a_i can be expressed by a Boolean function f.

A. Sequential Logic Clusters

A sequential logic cluster (SLC) is a structure on a FPGA that organizes LUTs and flip flops in sequential logic to accomplish the fast hardware implementation of the DBF form

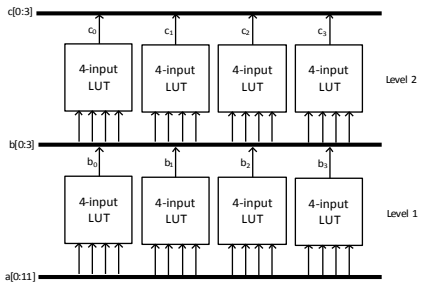


Fig. 2: $f_{compact}$ architecture example: combinational logic implementing the functions in (1c) and (1d).

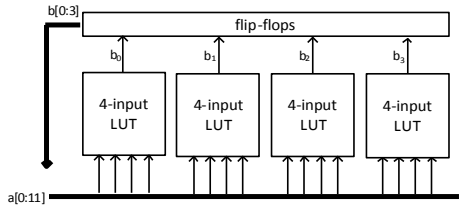


Fig. 3: $f_{compact}$ architecture example: sequential logic implementing the functions in (1c) and (1d).

of $f_{compact}$. The SLC shown in Figure 4 has m inputs, w LUTs and w flip flops. This design forms a cycle that can keep repeating sequentially, in which each cycle is equivalent to an iteration in $f_{compact}$. When it comes to the first cycle, a random seed of m bits is used as the primary inputs. The w LUTs select their inputs from the m bits of primary inputs after random shuffling. Each LUT randomly chooses k bits as its inputs, in Example I, k is equal to 4. Then in cycle 2, the w outputs generated by the w LUTs in cycle 1 are stored in the w flip flops, together with the m bits of primary inputs, becoming the cycle's inputs. The cycles can be repeated many times. This sequential architecture provides a compact and fast implementation of $f_{compact}$.

Consider the generation of such SLC structure, we can easily produce large amount of unrepeated SLCs by using different random shuffling and different contents of SRAM cells in the LUTs. As explained before, the randomly generated SLC structure directly represents a $f_{compact}$. Meanwhile, its corresponding f_{expand} can also be easily generated by the SLC. The method is to test all the possible input vectors to the SLC and if the output is 1, then add the input vector as a product to the sum of products while if the output is 0, then add the input vector as a sum to the product of sums. Therefore, both $f_{compact}$ and f_{expand} can be easily produced, which actually provides a very straightforward way to create the pairs of private key and public key in the DBF based public key communication protocol.

B. f_{expand} Synthesis Analysis

Now we consider the hardware implementation of f_{expand}. We use the Xilinx ISE to synthesis f_{expand} and compare the resources it takes with $f_{compact}$. For a SLC with a given number of inputs and cycles, we generate the corresponding f_{expand}, then convert it to a netlist and synthesis it to acquire the number of LUTs f_{expand} requires in order to be implemented on the FPGA. We conclude from Table II that f_{expand}

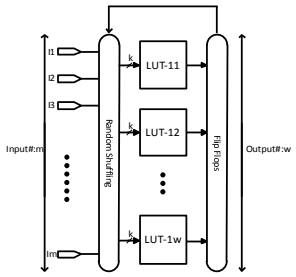


Fig. 4: Architecture of sequential logic cluster

requires many more LUTs than $f_{compact}$ and the difference keeps growing with the increase in the number of inputs and the number of cycles, which could easily reach the extent that f_{expand} costs so much to be implemented in hardware that it can only be simulated. In this way, the time difference between $f_{compact}$ (private key) and f_{expand} (public key) can be further expanded.

Cycle #	Input #	$f_{compact}$ LUT #	f_{expand} LUT #
4	8	8	81
5	10	10	396
6	12	12	1616
7	14	14	4353
8	16	16	13576
9	18	18	31155
10	20	20	98282

TABLE II: Average synthesis resources compared between DBF form $f_{compact}$ and form f_{expand}. The Input # does not have to be twice as the circle #, we just set it here as an example. The tests are based on the Spartan-3 XC3S50-5 FPGA and synthesized using the Xilinx ISE.

C. $f_{compact}$ and f_{expand} Comparisons

Based on the above discussion, Table III shows the comparisons between $f_{compact}$ and f_{expand}. Both of them can be simulated, but only $f_{compact}$ can be implemented by the sequential logic on an FPGA with a reasonable amount of resources. As implementation is generally faster than simulation, we choose to implement $f_{compact}$ rather than to simulate it. We see that with the scale of 20 primary inputs and 10 iterations, the implementation of $f_{compact}$ takes only 75.9ns while the simulation of f_{expand} takes more than 1.5×10^7ns. The time difference reaches 2.1×10^5.

The sequential logic architecture based on configurable logic blocks (CLBs) on FPGA provides a very intuitive and simple way to implement $f_{compact}$, while for f_{expand}, the implementation on FPGA takes too many resources, requiring it to be simulated. The time and efficiency difference between $f_{compact}$ and f_{expand} on FPGAs, which would be utilized for security purposes, offers us a most strong reason to choose FPGA as an ideal platform. For a system that requires high security, it is suggested to use SLCs with 64 inputs. In this case, although the public key size grows to be large, consider Protocol 1 that neither the computation expense of the private

	$f_{compact}$	f_{expand}
Implementation (fast)	yes	no
Simulation (slow)	yes	yes
Time expense (ns) (20 inputs, 10 iterations)	implementation: 75.9 ± 8.42	simulation: $(1.57 \pm 0.26) * 10^7$

TABLE III: Comparisons between $f_{compact}$ and f_{expand}. The time expense is in 95% interval confidence.

key holder nor the computation expense of the public key user grows much. Because private key holder only needs to configure the FPGA to run the SLC with 64 inputs for 32 cycles while the public key user only requires one product or one sum of each function in public key to compute the result. Therefore, only the attacker needs to simulate the whole huge public key which takes so much time and space that makes it impossible.

D. Time Estimation

We estimate the decryption time difference between the private key holder and the attacker in Protocol 1. The private key holder uses the SLC with 64 inputs and 32 cycles to implement $f_{compact}$ and the attacker simulates the corresponding f_{expand}. For the computation of $f_{compact}$, the implementation time is tested on the Spartan-3 XC3S50-5 FPGA to be approximately $239ns$. For the computation of f_{expand}, the simulation time is too long that can only be estimated. It is obvious that the simulation time is proportional to the size of the f_{expand}. According to Table I, we assume that the size of f_{expand} at least increase by 2.5 times with the increase of 1 iteration and 2 inputs. As a consequence, we suppose that the simulation time for f_{expand} also grows in such way. Therefore, for a SLC with 64 inputs and 32 iterations, the simulation time can be estimated as $1.57 \times 10^7 \times 2.5^{22} = 8.92 \times 10^{15}(ns)$. We further suppose that the number of rounds N in public key communication is 10^3, therefore, the time for private key calculation is $239 \times 10^3 = 2.39 * 10^5(ns)$ while the time for public key calculation is $8.92 \times 10^{15} \times 10^3 = 8.92 * 10^{18}(ns) \approx 283\ years$, which is not acceptable. Note that N is a flexible parameter that can be adjusted according to the size of public key.

E. Power Estimation

We estimate the power consumption of the encryption and decryption steps of the DBF-based public key communication protocol based on the same model in the time estimation part. For the decryption part, to implement a 64 inputs SLC, we need 64 LUTs. The static power value of Xilinx FPGA is approximately $24\mu W/CLB$. Suppose each CLB contains 4 LUTs, then 16 CLBs are required with a total power of approximate $384\mu W$. With 10^3 rounds of operations, the total clock cycle that decryption requires is 3.2×10^4. For the encryption part, according to the protocol, the public key user only needs to calculate one product or one sum of each function in f_{expand}, assume we use the Core i5-680 processor with $73W$ and $3.6GHz$ to simulate that product or sum. For each round only one clock cycle is required and 10^3 clock cycles are required with 10^3 rounds. We therefore have an energy consumption estimation for encryption and decryption through equation 2. Table IV shows that both the encryption and decryption steps of the DBF-based public key communication protocol are extremely low-energy.

$$Energy = Power \times ClockCycle\ \#/Frequency \quad (2)$$

Operation	frequency(GHz)	Clock Cycle #	Energy(μJ)
Decryption	0.13	$3.2 * 10^4$	9.18×10^{-2}
Encryption	3.6	10^3	20.3

TABLE IV: Public key communication: encryption and decryption energy consumption.

IV. Attacks

In this section, we analyze DBF's resistance against different potential security attacks. The simulation is conducted on the SLC with 64 primary inputs, 32 iterations, and 1,000,000 input vectors.

A. Avalanche Criterion

In this attack, the attacker tries to predict outputs by knowing the outputs having similar inputs. In a cipher with good diffusion, if one bit of the inputs is changed, then the outputs should change completely and in an unpredictable manner. So, we test the hamming distance between the output vectors by changing one bit of the input vectors every time. Ideally, the result should be in the form of a polynomial distribution with the peak on the half of the number of outputs. The result in Figure 5(a) shows an almost perfect polynomial distribution of hamming distance, indicating DBF is highly resilient against this type of attack.

B. Frequency Prediction

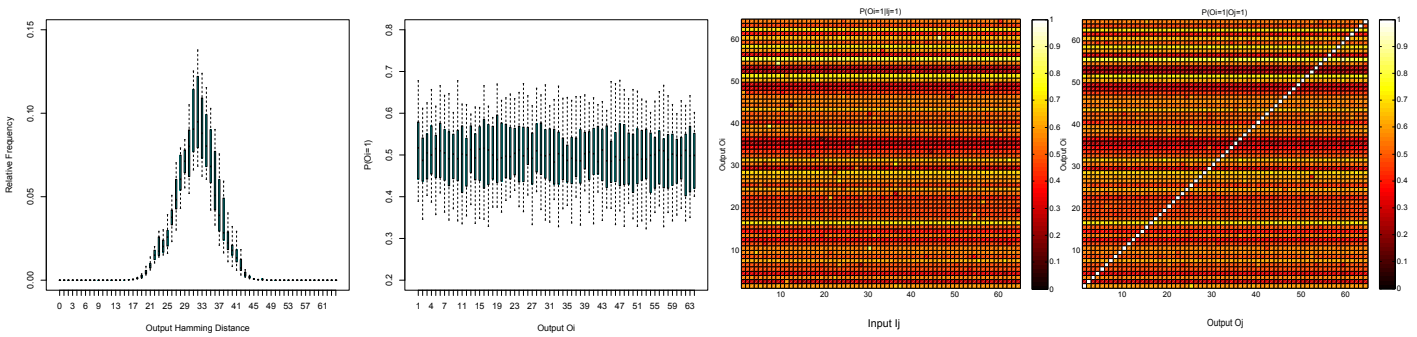
In this attack, the attacker collects the output data from previous DBFs and builds a probability distribution for each output being a particular value. Ultimately, the attacker tries to predict each output O_i by using the distribution. The goal of the attacker is to predict $P(O_i = c)$, where c = 0 or 1. The ideal situation is that an output is 0 or 1 with both probability 0.5. Figure 5(b) shows that the mean value of the probability that each output bit is equal to 1. The probability is always close to 0.5, indicating the high unpredictability of the structure.

C. Conditional Correlation

Another type of attack is that the attacker tries to look into the correlation between an output bit O_i and an input bit I_j or another output bit O_j. The goal of the attacker is to predict $P(O_i = c_1 | I_j = c_2)$, $P(O_i = c_1 | O_j = c_2)$, c_1, c_2=1 or 0. For example, if the attacker observes that output i is 1 a large majority of the time that input j is 1, and if the current input j is 1, then he can guess that output i will be 1 with higher likelihood. The ideal situation is that all these probabilities are 0.5. Figure 5 (c) and (d) respectively shows the colormap of conditional probability $P(O_i = 1 | I_j = 1)$ and $P(O_i = 1 | O_j = 1)$. We see that these probabilities are close to 0.5, suggesting low potential for prediction.

V. Performance Comparisons

Table V shows the comparisons for DBF based public key communication with traditional block cyphers and RSA with respect to area, delay, and energy in FPGAs. The DBF in the table utilizes one SLC with 64 primary inputs and 32 iterations, and we suppose that the number of rounds N in the protocol is 10^3. We can obviously conclude from the table that DBF, as a security primitive, owns the implementation of ultra low energy that is competitive with the traditional security key block cyphers and outperforms the RSA with at least three orders of magnitude.



(a) (b) (c) (d)

Fig. 5: (a) The distribution of output hamming distance for avalanche effect (the error bar shows the distribution of max, 75%, mean, 25% and min), (b) Probability that an output bit is equal to 1, (c) Conditional Probabilities between Output bits O_i and input bits I_j, (d) Conditional Probabilities between Output bits O_i and other output bits O_j.

Design	Flip Flops	LUTs	Area (Slices)	Maximum Delay (ns)	Clock Cycles	Energy (μJ)	Block Size (bits)	Throughput (Mbps) at f_{max}	Throughput/Energy (Mbps/μJ)	Device
Present[1]	114	159	117	8.78	256	3.16×10^{-3}	64	28.46	9.01×10^{3}	xc3s50-5
HIGHT[1]	25	132	91	6.12	160	1.07×10^{-3}	64	65.48	6.12×10^{4}	xc3s50-5
AES[1]	338	531	393	14.21	534	3.58×10^{-2}	128	16.86	4.71×10^{2}	xc3s50-5
RSA[19]	1870	2811	1553	7.62×10^{3}	907	128.80	-	0.15	1.16×10^{-3}	xc3s500e
RSA radix-2[19]	7564	11496	6282	8.21×10^{3}	1058	654.80	-	0.12	1.83×10^{-4}	xc2v6000
RSA radix-4[19]	9944	14907	8328	4.23×10^{3}	560	236.73	-	0.43	1.82×10^{-3}	xc2v6000
DBF	64	64	32	7.47	32000	9.18×10^{-2}	-	267.74	2.92×10^{3}	xc3s50-5

TABLE V: Comparisons for DBF based cryptography with the traditional block cyphers and RSA. The results for Present, HIGHT and AES are cited from [1], the results for RSA are the parts of multiplication modular and are cited from [19], the results for DBFs are tested on the Spartan-3 XC3S50-5 FPGA and generated by the Xilinx ISE Design Suite 14.3.

VI. Conclusion

We have developed the DBF, an ultra low power cryptographic primitive that enables security protocols such as public key communication that requires low energy consumption for all legitimately involved parties. The primitive uses two forms of Boolean functions, one is created in a compact format and has a low-energy FPGA implementation while the other is ultra complex for both implementation and simulation. Simulation results show that DBF is resilient to a wide variety of security attacks and the energy consumption is more than three orders of magnitude lower than the traditional public key cryptographic implementations (e.g., RSA).

References

[1] P. Yalla, J-P. Kaps, "Lightweight cryptography for fpgas," *Reconfigurable Computing and FPGAs, 2009. ReConFig'09. International Conference on.* IEEE, 2009.

[2] J-P. Kaps, *Cryptography for Ultra-Low Power Devices*, unpublished thesis (PhD), Worcester Polytechnic Institute, 2006.

[3] M. Majzoobi, F. Koushanfar, M. Potkonjak, "Lightweight secure PUFs," *ICCAD 2008*, pp. 670-673, 2008.

[4] R. Pappu, B. Recht, J. Taylor, N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[5] B. Gassend, D. Clarke, M. van Dijk, S. Devadas, "Silicon physical random functions," *ACM Conference on Computer and Communications Security*, pp. 148–160, 2002.

[6] S. Devadas, V. Khandelwal, S. Paral, R. Sowell, E. Suh, T. Ziola, "Design and Implementation of PUF-Based Unclonable RFID ICs for Anti-Counterfeiting and Security Applications," *IEEE RFID*, 2008.

[7] S. Wei, M. Potkonjak, "Wireless security techniques for coordinated manufacturing and on-line hardware trojan detection," *WISEC*, pp. 161-172, April 2012.

[8] J. Zheng, M. Potkonjak, "Securing Netlist-Level FPGA Design through Exploiting Process Variation and Degradation," *FPGA*, pp. 129-139, February 2012.

[9] N. Beckmann, M. Potkonjak, "Hardware-based public-key cryptography with public physically unclonable functions," *Information Hiding Conference*, pp. 206–220, 2009.

[10] U. Rührmair, "SIMPL systems, or: can we design cryptographic hardware without secret key information?" *International Conference on Current Trends in Theory and Practice of Computer Science*, vol. 6543, pp. 26–45, 2011.

[11] S. Meguerdichian, M. Potkonjak, "Security Primitives and Protocols for Ultra Low Power Sensor Systems," *IEEE SENSORS*, pp. 1225-1228, October 2011.

[12] S. Meguerdichian, M. Potkonjak, "Using standardized quantization for multi-party PPUF matching: Foundations and applications," *Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on*. IEEE, 2012.

[13] M. Majzoobi, F. Koushanfar, M. Potkonjak, "Techniques for Design and Implementation of Secure Reconfigurable PUFs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 2, no. 1, 2009.

[14] J. Rajendran, G. S. Rose, R. Karri, M. Potkonjak, "Nano-PPUF: A Memristor-Based Security Primitive," *VLSI (ISVLSI), 2012 IEEE Computer Society Annual Symposium on*, pp. 84-87, 2012.

[15] S. Meguerdichian, M. Potkonjak, "Device Aging-Based Physically Unclonable Functions," *Design Automation Conference (DAC)*, pp. 288-289, June 2011.

[16] J. B. Wendt, M. Potkonjak, "Nanotechnology-Based Trusted Remote Sensing," *IEEE SENSORS*, pp. 1213-1216, October 2011.

[17] M. Majzoobi, F. Koushanfar, M. Potkonjak, "Testing Techniques for Hardware Security," *IEEE International Test Conference*, pp. 1-10, 2008.

[18] S. Meguerdichian, M. Potkonjak, "Matched public PUF: ultra low energy security platform," *IEEE/ACM ISLPED*, pp. 45–50, 2011.

[19] E. Oksuzoglu, E. Savas, "Parametric, secure and compact implementation of RSA on FPGA," *Reconfigurable Computing and FPGAs, 2008. ReConFig'08. International Conference on*. IEEE, 2008.