The Potential of Reconfigurable Hardware for HPC Cryptanalysis of SHA-1

Alessandro Cilardo
Computer Science Department, University of Naples Federico II,
via Claudio 21, 80125 Napoli, Italy, Email: acilardo@unina.it

***Abstract*—Modern reconfigurable technologies can have a number of inherent advantages for cryptanalytic applications. Aimed at the cryptanalysis of the SHA-1 hash function, this work explores this potential showing new approaches inherently based on hardware reconfigurability, enabling algorithm and architecture exploration, input-dependent system specialization, and low-level optimizations based on static/dynamic reconfiguration. As a result of this approach, we identified a number of new techniques, at both the algorithmic and architectural level, to effectively improve the attacks against SHA-1. We also defined the architecture of a high-performance FPGA-based cluster, that turns out to be the solution with the highest speed/cost ratio for SHA-1 collision search currently available. A small-scale prototype of the cluster enabled us to reach a real collision for a 72-round version of the hash function.**

I. Introduction

Modern FPGAs can be used to build application-specific high-performance computing (HPC) machines at costs that are several orders of magnitude lower than standard HPC platforms. This has some important implications for those applications where the (un)availability of suitable computing resources is an essential underlying assumption. Cryptographic algorithms are perhaps the most remarkable example, since most of them are based on some *hard problems*, supposed to be intractable with ordinary computing resources. Cost is not the only factor that may give reconfigurable technologies a special role for cryptanalysis. In fact, reconfigurable hardware, by its nature, can be used "interactively", changing the hardware design over time more than once. So, not only can FPGAs be used for the cryptanalytic computation in itself, but also for supporting the study of the problem, enabling large-scale computation efforts just aimed at algorithm and architecture exploration.

This paper, in particular, addresses the possibility of breaking the most important cryptographic hash function currently in use, SHA-1, a hot topic in today's cryptanalytic research. The background in hardware cryptanalysis and attacks against cryptographic hash functions is reviewed in Section II, while a deeper presentation of the SHA-1 algorithm and the cryptanalysis methods used in this work is provided in Section III. Starting from the current state-of-the-art, Section IV then investigates innovative approaches, inherently based on hardware reconfigurability, enabling extensive algorithm and architecture exploration for the cryptanalysis of SHA-1 and identifying new techniques for building effective attacks. In addition, it demonstrates how static and dynamic circuit specialization can play a key role to maximize the performance of the hardware system by customizing it for specific parameters. These opportunities are extensively exploited by a set of software tools we developed to automatically generate highly optimized HDL code for specific input collision search parameters. To demonstrate our approach, we have developed a prototypical FPGA cluster, built on top of a highly optimized dedicated unit for SHA-1 collision search, described in Section V. A quantitative evaluation of performance and implementation efficiency confirms the impact of the different techniques employed. As shown in Section VI, in fact, the FPGA-based architecture can reach the same levels of performance as an optimized collision search application run on a massively parallel HPC cluster, while costing two orders of magnitude less. By using a small-scale prototypical cluster, made of only 20 very low-cost FPGAs, we were able to find an actual collision for a 72-round version of SHA-1, outperforming the best achievement presented in the technical literature.

978-3-9810801-7-9/DATE11/©2011 EDAA

II. Background

Cryptanalysis usually requires massive computations and often relies on special-purpose hardware solutions exhibiting much better performance/cost ratios than off-the-shelf computers. While there would be countless examples in the literature of hardware acceleration for cryptanalysis, there are few contributions that deliberately use reconfigurable devices as the underlying technology. Some commercial clusters relying on FPGAs, for example, have been recently suggested for use in cryptanalytic applications, including Copacobana, made of 120 Spartan-3 1000 or Virtex-4 SX35 FPGAs [1], and its successor Rivyera, made of 16 to 128 Spartan-3 5000 FPGAs [2]. Other contributions in the literature focus on low-level aspects. For instance, it has been shown how some FPGA-specific resources can accelerate RSA encryptions [3] as well as attacks on RSA [4].

Among the numerous works on hardware-accelerated cryptanalysis, very few target collision search for cryptographic hash functions, in spite of the serious threats to security created by vulnerabilities in hash functions such as MD5 [5]. Reference [6], in particular, presents a special-purpose microprocessor to speedup collision search for MD4-family hash functions. The core has minimal area requirements but it basically uses a software-like implementation of the collision search algorithm and, in fact, it only shows significant improvements if implemented as an ASIC. Many works exist, on the other hand, on theoretical or software-based attacks to SHA-1. They are especially relevant here for comparisons of attack performance and costs. After some early studies by Chabaud and Joux [7] on SHA-1's predecessor SHA-0, based on a differential approach, and by Biham et al. [8], Wang et al. [9] showed in 2005, for the first time, a method to find a collision for SHA-1 with a theoretical complexity lower than the bound of 2^{80} of a simple birthday attack, namely 2^{69}. De Cannière et al. [10] then described a way to automatically find complex Non-Linear characteristics and used it to determine a two-block colliding message pair for a weakened 64-round version of SHA-1. A collision for a 70-round version of

SHA-1 was presented by the same researchers in [11] and an equivalent result was obtained by T. Peyrin in [12]. Finally, Cilardo et al. [13] presented a study of vulnerabilities in the SHA family, namely the SHA-0 and SHA-1 hash functions, based on a high-performance computing application run on a massively parallel cluster. They were able to identify the first collision for a 71-round version of SHA-1.

III. The SHA-1 hash function

Issued by NIST in 1995 as a Federal Information Processing Standard [14], SHA-1 is the most popular hash function currently in use for cryptographic applications. The hash function SHA-1 takes a message of length less than 2^{64} bits and produces a 160-bit hash value. The input message is padded and then processed in 512-bit blocks in the Merkle-Damgård iterative structure. Each iteration invokes a so-called compression function which takes a 160-bit chaining value and a 512-bit message block and outputs another 160-bit chaining value. The initial chaining value (called IV) is a set of fixed constants, and the final chaining value is the hash of the message. The compression function of SHA-1 works as follows. For each 512-bit block of the padded message, divide it into sixteen 32-bit words, $(W_0, W_1, \ldots, W_{15})$. The message words are first *expanded* as follows: for $i = 16 \ldots 79$

$$W_i = (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) << 1 \qquad (1)$$

where the '$<< x$' notation is used for a left rotation of x bits. The 160-bit chaining value is stored into five internal 32-bit variables, called A, B, C, D, E. The expanded message words W_i are then processed in 80 rounds, divided into four groups of 20 consecutive rounds. Each of these 80 steps applies the following *round function*: For $i = 1 \ldots 80$

$$A_i = (A_{i-1} << 5) + f_i\,(B_{i-1}, C_{i-1}, D_{i-1}) + E_{i-1} + W_{i-1} + K_i$$
$$B_i = A_{i-1} \quad C_i = B_{i-1} << 30 \quad D_i = C_{i-1} \quad E_i = D_{i-1} \qquad (2)$$

where the '$+$' symbol denotes the integer addition performed modulo-32. Each group of 20 rounds uses a different Boolean function f_i and constant K_i, as summarized in the following table.

round	**Boolean function** $f_i(x, y, z)$	**constant** K_i
$0 - 19$	$IF : (x \cdot y) + (\overline{x} \cdot z)$	`0x5A827999`
$20 - 39$	$XOR : x \oplus y \oplus z$	`0x6ED6EBA1`
$40 - 59$	$MAJ : (x \cdot y) + (x \cdot z) + (y \cdot z)$	`0x8FABBCDC`
$60 - 79$	$XOR : x \oplus y \oplus z$	`0xCA62C1D6`

The chaining value $IV = (A_0, B_0, C_0, D_0, E_0)$ for the first application of the compression function is defined by the standard as (`0x67452301`, `0xEFCDAB89`, `0x98BADCFE`, `0x10325476`, `0xC3D2E1F0`).

A. Cryptanalysis of SHA-1

The essential idea behind the attack to SHA-1 is to constrain the values of the messages and registers $A \ldots E$ in order to reach a collision with a certain probability. Such set of bit-level constraints on the difference of two messages is called "differential characteristic" (an example is shown in Figure 1). A differential characteristic is comprised of two sections. The $\mathcal{W}$ part defines the constraints imposed to the bits of the two messages in each position, while the $\mathcal{A}$ part contains the constraints imposed to the internal registers $A \ldots E$ during the hashing process. In fact, the characteristic only considers the A register, since the remaining four variables $B \ldots E$, and related constraints, can be easily obtained from A by simple rotations and round shifts (e.g. $D_i = A_{i-3} << 30$). The constraints are expressed by a set of symbols: '1' and '0' indicate that both bits in the two messages must take on the values 1 and 0, respectively, 'u' and 'n' indicate a signed difference ($1/0$ or $0/1$, respectively), 'x' an unspecified difference, '$-$' two unspecified equal values.

```
round        A(i)                 W(i)        Dw(i) Pu(i)  N(i)
   /--..-----32bit----/ /--..-----32bit----/ .   (log2)  .
-4: 00..00001111000011                       .     .     .
-3: 01..10100011101100                       .     .     .
-2: 01..11100111111010                       .     .     .
-1: 11..01010111000100l                      .     .     .
00: 01..010001100000001 u1..11-1-010u0n10un  3   0.00 2^ -9.16
01: n1..00-0-0111uu110u 01..0----001000n010  7  -4.19 2^ -2.16
02: 1n..0--u-n10011u1u0 01..-------1n1011nu  11 -6.06 2^4.65
03: 11..u----0-uu0uuu0u nn..--00---1uun10u1  10 -9.19 2^8.59
04: n0..1un0u1000011111 n1..0----0-000u0010  6  -6.00 2^5.4
05: 10..u110111110000n0 uu..-------0nn010u1  9 -10.00 2^8.4
06: 11..110000n100011uu un..---0-0-0u1011u0  5  -5.00 2^3.4
07: u1..nu00-01n0n01n1u 11..11-------uu0001  8  -7.00 2^6.4
08: 11..1n11-01011u1u01 nu..--1--111u1111u0  6  -4.00 2^5.4
09: 10..0-1n-111u001un0 11..001----1nu11000  7  -6.00 2^9.4
10: 01..1nu0-000001n10n nn..-------0n10110n  9  -4.19 2^11.4
11: u1..-0-1nu-1u000un0 01..---011-1u0n0010  11 -6.94 2^18.21
12: u0..---u000u0nu0111 nu..-------0u01100u  13 -5.00 2^24.27
13: 10..-----1001u1u1uu nu..--------unu00u0  15 -6.00 2^34.27
14: 00..-----1--10u1u11 u0..--------11101nn  14 -3.27 2^42.27
15: u0..--------0011001 nn..---------1u00u0  14  0.00 2^53
16: -0..---------1n101u 0n..-------1n01011u   0 -2.00 2^53
17: u1..-------010-111- nn..-------01nn10u0   0 -1.00 2^51
18: -1..----------100-- 00..--------100001n   0  0.00 2^50
19: --..----------10-- nu..------1011010u1   0 -2.00 2^52<=AP
20: --..----------1-n- nu..------01u0111u1   0 -2.00 2^50
21: --..-------------n- 1n..--------u0101n0   0  0.00 2^48
       ......      ......       ......         ......
68: --..----------u--- --..----00n0-1xn-ux   0 -3.00 2^11
69: --..----------n--- --..-----1u0-u-xx-u   0 -3.00  2^8
70: --..---------u----- --..----n0-1--nxxx-   0 -4.00  2^5
71: --..----------n---- --..-----u0-0x--u--   0 -1.00  2^1
72: --..---------------                       0        1
```

Fig. 1. An example of differential characteristic

As a preliminary work, we developed some support tools for the generation of the characteristic, based on both existing and original techniques. We will not give the details of these tools here, as they would be out of the scope of the paper. For each round i, the support tools evaluate the probability that the subsequent A_{i+1} complies with the characteristic, given that the current word W_i and the previous five $A_i \ldots A_{i-4}$ (i.e., registers $A_i \ldots E_i$) do as much. This probability is called $P_u(i)$. For the first sixteen W_i, which are in fact a part of the message to be hashed, some bits are not constrained by the characteristic (those indicated as '$-$') so that they can be controlled by the attacker. They are called *degrees of freedom* and their number for each round is denoted as $D_w(i)$. Clearly, $D_w(i) = 0$ for $i \geq 16$ due to the expansion function (1) for computing words W_i. The collision search proceeds by setting the degrees of freedom and evaluating pairs of messages whose difference complies with a given characteristic.

In order to evaluate the performance of the attack, it is important to estimate the expected number of executions for each round i, denoted as $N(i)$. This parameter can be computed from $P_u(i)$ and $D_w(i)$ starting from the bottom of the characteristic, by using the recursive relationship $N(i+1) = N(i) \cdot P_u(i) \cdot 2^{D_w(i+1)} \Rightarrow N(i) = N(i+1) \cdot 2^{-D_w(i+1)} / P_u(i)$ with the initial value of $N(i+1)$, i.e. the number of execution of the last round (72 for the characteristic in the figure) equal to 1, since we will stop the search process as soon as we reach the last round for the first time. Since the execution of

each round, i.e. the computation of all relationships (1) and (2) for W_i and $A_i \dots E_i$ and the corresponding register updates, can be performed in parallel, possibly in a single clock cycle for a hardware implementation, we call this set of steps an *Elementary Operation* (EO), and the expected total number of round executions, i.e. the summation of parameters $N(i)$ for all rounds i, the *Mean number of EOs to Collision* (MEOC).

Due to the exponential complexity of the search process, it is essential to find out efficient ways to enumerate all available message pairs in the search space, possibly by identifying "more probable" message pairs and by pruning as early as possible large subspaces not containing collisions. An improvement to the algorithm is provided by the use of *Auxiliary Paths* (APs, see [15]). An AP is a set of bits in the message pair which, if flipped, produce another message pair satisfying the characteristic until a certain round r_{AP}. Thus, once a message pair is tested to be compliant to the characteristic until round r_{AP}, it is possible to generate another message pair that will certainly be compliant in that round. In effect, since r_{AP} is located late in the round sequence, when the probability of execution has already decreased by several orders of magnitude, the fork produced by an AP virtually doubles the chances of finding a collision, i.e. it halves the MEOC. Clearly, a number P of different APs allows the generation of $2^P - 1$ new "good" message pairs from r_{AP} onwards, and a corresponding improvement for the MEOC. The characteristic in Figure 1 has two APs that can be applied at round $r_{AP} = 19$. All $N(i)$ values above the fork, computed with the formula given earlier, are thus corrected with a decrement by 2. Unfortunately, taking into account an AP also involves a cost related to saving the intermediate search state and restoring it after the previous execution branch. This overhead makes the EO, or round, where the AP fork occurs have a time cost larger than other EOs. In general, furthermore, especially for parallel implementations, similar forks –and related overheads– take place at some specific rounds for dividing the search space among different processors. As a consequence, the execution times in terms of clock counts $C(i)$ for each round i can be different and, thus, the MEOC is not necessarily proportional to the total clock count for a collision. The *Mean Clock Count to Collision* (MCCC), rather, should be computed as the weighted sum of the EO counts $N(i)$, i.e. $MCCC = \sum N(i) \cdot C(i)$. Since the MEOC, including the APs' contribution, is specific to a given characteristic, while the times $C(i)$ depend on the different architectural optimizations employed, a good metric for the quality of an implementation is the $MEOC/MCCC$ ratio, called here *EO per clock cycle*, or EOC. For a single, basic core performing a sequence of EOs on W_i pairs, the ideal bound for EOC is 1.

IV. Hardware reconfiguration for the cryptanalysis of SHA-1

Reconfigurable hardware, by its nature, can be used "interactively", changing the hardware design over time more than once. So, not only can FPGAs be used for the cryptanalytic computation in itself, but also for supporting the study of the problem, enabling large-scale computation efforts just aimed at algorithm and architecture exploration. We extensively exploited this potential, as described below, to investigate new effective search techniques. In addition, we found out that both static and dynamic circuit specialization, enabled by reconfigurable devices, can play a key role to maximize the performance and minimize the hardware complexity of the system, automatically specializing it for a given input characteristic.

a) Algorithm and architecture exploration: As a first step in our study, we employed reconfigurable hardware for extensively experiment with new algorithmic techniques. Two remarkable opportunities were identified by this approach:

1) We developed an FPGA-based temporary design to explore the behavior of higher rounds in the differential characteristic, namely those beyond round 60, that are normally executed very rarely due to the exponentially decreasing value of probability. We found that some bit-level constraints in the last rounds, seemingly independent of the previous constraints, are in fact indirectly influenced by some bit patterns including the degrees of freedom in the sixteen message words, due to the message expansion function for W_i. These *inter-bit constraints*, resulting in a degraded actual probability of success (normally halved for each constraint) had not been detected before, because they impact the behavior of higher, i.e. very rare, rounds. The temporary search hardware was designed to bypass the middle phase of the collision search process and, albeit not able to find an actual collision, it could detect the misalignments in the actual behavior at the higher rounds. It turned out that the differential characteristic, as it is normally defined, is not suitable to capture the effect of inter-bit constraints, essentially because it only expresses bit-level constraints. Moreover, the additional constraints depend *dynamically* on the specific bit pattern of some degrees of freedom chosen during message enumeration. To cope with this problem, we devised a technique where the differential characteristic -and the corresponding hardware- is dynamically changed during message enumeration to mask the inter-bit constraints. More precisely, at design-time we analyze the characteristic and express all inter-bit constraints in a system of Boolean linear equations, reduced by Gaussian elimination is such a form as to concentrate all independent variables (some degrees of freedom in the message words) as high as possible in the first sixteen words. As the independent variables are set during message enumeration, the corresponding constraints in the characteristic are set accordingly by back-substitution in the system. In practice, this process only requires the computation of a sequence of XOR operations and, importantly, it does not affect the execution time significantly since it only happens before round 15, i.e. for rounds that are orders of magnitude less frequent than the subsequent ones.

2) A second technique that was successfully investigated is called *constraint relaxation*. Basically, we experimented with new constraints in the characteristic, different than the bitwise conditions imposed by the $\mathcal{A}$ part. The essential idea was to employ integer differences rather than bitwise differences. Under some circumstances, this may deteriorate the probability $P_u(i)$ less than expected, while enlarging the set of tries that satisfy the conditions, resulting in an improved overall MEOC. A difficult problem was to identify such situations, if any, and define the actual positions and parameters for the constraint relaxation. To this aim, we developed an ad-hoc message enumeration component designed to skip or reinterpret some part of the compliance verification process in order to identify the locations where the relaxation could pay off. The hardware-supported analysis led us to identify rounds 32 and $64 - 72$. The actual relaxation consists in replacing the bitwise XOR between the A and A' registers for the two messages in the pair with their integer absolute difference $|A - A'|$.

Of course, there is no space here to present the different

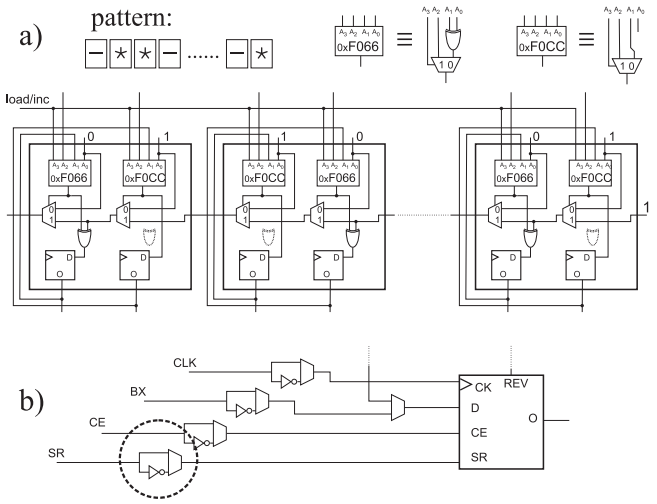

Fig. 2. Some optimizations enabled by hardware reconfiguration

variants of the hardware system used for algorithmic exploration, although they all share the same high-level architecture presented in Section V for the SHA-1 collision search cluster.

b) Parameter-dependent optimizations in the static design: Reconfigurable technologies enable another important design approach that can be especially relevant for cryptanalysis: they allow the *ad-hoc* generation of highly optimized systems tailored on specific input parameters. We identified numerous situations for the collision search process where this approach has a considerable impact on the implementation efficiency. Some examples include the logic for the fast flipping of Auxiliary Path bits, the XOR network for applying inter-bit constraints, the logic for the compliance check and constraint relaxation, the implementation of "segmented incrementers" for a more efficient message enumeration, and many others. Just as one example (out of many actually implemented), we provide some details on segmented incrementers. As explained above, the search process works basically by enumerating all the combinations for the '$-$' symbols in the characteristic (see Figure 1). This cannot be done by using normal counters in a straightforward way, since bits are in general not consecutive, and may compromise the use of carry propagation logic for fast increments. However, for a typical configurable element, e.g. a Xilinx Spartan3 slice like those depicted in Figure 2, a carefully optimized (and, of course, characteristic-dependent) configuration allows us to pack in a single group of consecutive FPGA Look-Up Tables (LUTs) and Flip-Flops all the bits in their order, exploiting carry propagation logic for the counting operation, skipping unaffected bits without interrupting the carry chain, and –in addition– embedding a multiplexer for load operations from the outside. For the example pattern "$-\star\star-\ldots-\star$", where '$-$' symbols denote the bits to be enumerated and '$\star$' the constant values to be skipped, Figure 2.a) shows the appropriate configuration of LUTs and surrounding logic. Since this kind of low-level, input-dependent optimizations involves nearly all components of the design, the impact on both hardware complexity and circuit delay can be considerable. In practice, it roughly leads to halved footprints on the FPGA for the single elementary core, and hence doubled parallelism and computational power.

Another important example of input-dependent system customization is related to hardware replication for fast EO executions. Based on the actual probability values inferred from the characteristic, we resort to replication only for those rounds whose $N(i)$ is above a certain threshold, i.e. only when an improved $C(i)$ can significantly increase the EOC towards the ideal bound of 1. This is an important input-dependent optimization, that can have a dramatic impact on the overall execution time, as shown in Section V-A.

For a given input characteristic, the above optimizations determine statically the design to be implemented. To support the specialization of the system, thus, we have developed a set of tools for the automated generation of HDL code, taking into account the input characteristic, its APs, inter-bit constraints, constraint relaxation, low-level optimizations, etc. Needless to say, this customization of the hardware system based on specific input parameters is an inherent advantage of FPGAs, that would be impossible for an approach based on ASICs.

c) Dynamic updates of wired logic: Finally, we identified some situations where *dynamic* reconfiguration of some parts of the system can lead to higher speed and/or improved hardware complexity. Since, in general, dynamic reconfiguration requires a non-negligible time, its use is justified only when it occurs with a medium/coarse temporal granularity. Again, there would be many examples where it can pay off, including the cases where the logic to be applied depends on some previous settings made at run-time, e.g. for inter-bit constraints. Another example of such situations is related to register re-initialization, where some value set earlier in a register, and then overwritten, needs to be used again (e.g., during message enumeration, before entering a new branch in the search tree from a certain node). Typically, we would need an additional register or memory to store the value while all the branches below are explored. A possibility based on dynamic reconfiguration, on the other hand, would consist in using the Set/Reset (SR) configurable signal available for Flip-Flops in most FPGAs (see Figure 2.b)). By controlling the behavior of the SR signal appropriately, any initialization value can be set in the flip-flops making up the bits of a register. The SR MUX, of course, cannot be controlled directly from the user design, as it is part of the FPGA configuration. The intermediate values of registers $A \ldots E$ for the relatively rare round 14, kept constant as the search goes down through round 15 and beyond, and then changed every time we go back to round 13, could be an example of a situation benefitting from re-initialization based on dynamic reconfiguration. Since there are ten such registers in a single SHA-1 collision core (320 bits), and many cores in a single FPGA, the technique may save an appreciable quantity of hardware resources, although in general less compared to the impact of the static input-dependent optimizations described in the previous paragraph. Incidentally, the Xilinx Spartan3 devices we used for our experiments do not support dynamic reconfiguration, so we did not implement this class of optimizations.

V. The FPGA cluster

Figure 3.a) summarizes the application flow for the automated generation of the HDL code from an input characteristic and the configuration of the FPGA cluster. Based on an iterative, three-phase refinement process, not described here, a set of software tools generate a differential characteristic like that in Figure 1. The characteristic is then analyzed by a module for the automated generation of HDL code (namely, VHDL), leading to the configuration of the different components of the cluster architecture. This automated generation applies all the architectural optimizations that, as described in the previous section, depend on the specific behavior of

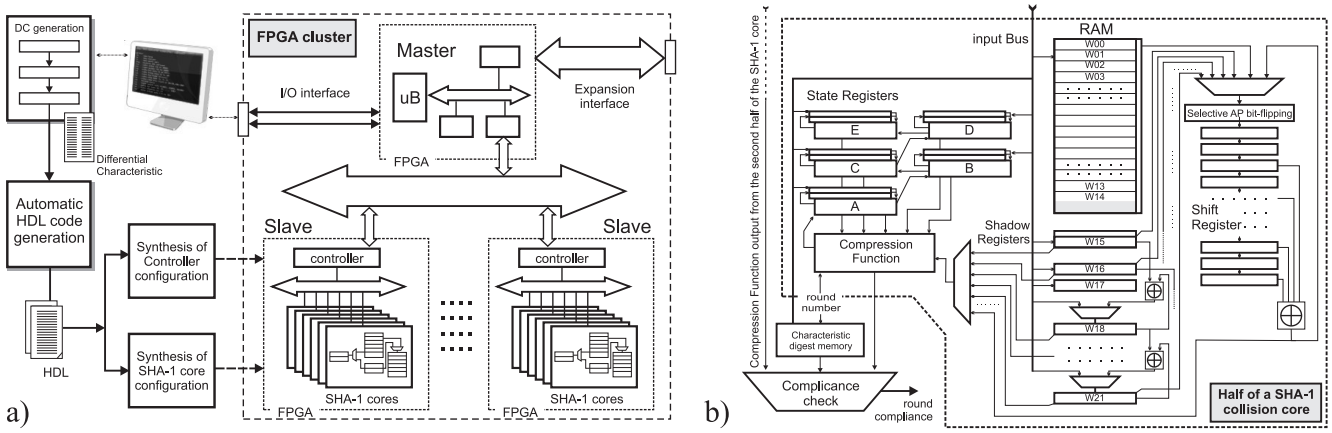

Fig. 3. a) Architecture of the application flow and FPGA cluster. b) One half of the SHA-1 collision core

the characteristic. The cluster architecture is made of three levels. A top-level Master node analyzes the first few rounds (e.g. the first thirteen), in order to produce more constrained characteristics, which are then sent to the Slave nodes. Most of the workload is concentrated below the first rounds, so that the concurrent jobs dispatched to the slaves achieve an almost complete parallelization of the search process. The jobs, moreover, involve very little communication overhead, since only the initial search state for each job and the possible colliding messages need to be exchanged over the bus. Within the slave node (a whole FPGA), a Controller component acts as a second level in the architecture. It makes further enumeration of intermediate rounds (e.g. on round 14) and distributes the remaining search workload to a set of *SHA-1 collision cores*, described in detail below, which constitute the third level in the architecture. We leave only round 15 for the enumeration on the third-level cores, as long as the available degrees of freedom ensure an execution time long enough to hide the synchronization overhead. This simplification allows further speed and area optimizations, enabling a high EOC with small-footprint cores.

For the current prototypical implementation of the cluster, we used a set of 20 inexpensive commercial off-the-shelf boards, namely the Digilent Nexys-2 boards, each equipped with a Xilinx Spartan XC3S1200E FPGA. The interconnection is made through an ad-hoc inter-board bus. The Master node is implemented as a microprocessor system based on the Xilinx MicroBlaze core, while the Slave nodes are completely custom-made. We paid much attention to the modular nature of the cluster, supporting an easy extension with additional hardware. The extension port allows the *hot-plug* of additional modules and the software-supported reorganization of the search partitioning as new modules are plugged in. The Controller also interact with an on-board non-volatile memory for the checkpointing of the search jobs. An I/O interface managed by the Master node allows the external user to interact with the cluster during the the search operation.

A. SHA-1 collision core architecture

The basic building block of the cluster consists of a core which is able to process sequences of EOs for a certain portion of the search space. Figure 3.b) shows a particular instance of the SHA-1 collision core, generated for the characteristic of Figure 1. We relied on many of the techniques identified in Section IV to determine the structure of the core. For time-critical rounds, we used selective hardware replication by means of suitable *shadow registers* (see the figure) with preloaded message expansion (possibly completed with a single XOR during enumeration). The first sixteen W_i words are stored in an inexpensive 32-bit LUT-based memory and accessed sequentially for filling in a shift register with a (time-consuming) serial-in load, but only for the relatively rare rounds beyond 21. The shadow registers and the shift register load circuitry contain some ad-hoc logic controlling the selective bit-flipping related to AP enumeration. A segmented incrementer is used for round 15. The registers $A \dots E$, making up a shift register, need one long- and one medium-term initialization value to be stored into two additional registers for each variable $A \dots E$, implemented as memory LUTs. The structure shown in Figure 3.b) is duplicated for each SHA-1 collision core to explore the behavior of a pair of messages concurrently. The output difference between the two halves of a core is used to verify the compliance with the characteristic. To save memory, only a *digest* of the whole 80-round characteristic is stored in each core, privileging the conditions on the more frequent rounds to limit false positives. The actual digest is defined, again, according to the input characteristic. The hardware for the compliance check is also responsible for constraint relaxation at the appropriate rounds.

As a result of these techniques, the EOC measured for the above SHA-1 collision core, specialized for the input characteristic of Figure 1, is very close to the ideal bound, precisely EOC$= 0.84$. Similar, or even higher, values were obtained for other characteristics.

A number of implementation-level optimizations were carried out for the target Spartan3 device. We made an extensive use of RLOC constraints and manual placement in order to obtain extremely regular layouts, enabling high efficiency in the use of the FPGA resources and decreased delays. To obtain the maximum level of compactness, we carefully balanced the use of flip-flops and LUTs used as 1-bit memory elements. The implementation results for both the SHA-1 collision core and the whole design are listed below. The SHA-1 core reduced footprint allows each of the 20 FPGAs in the prototypical cluster to host six cores.

	LUTs	**flip-flops**	**delay**
Controller Core	2539	2379	$12.0ns$
Single SHA-1 Core	2127	1979	$11.8ns$
Slave Node	15301	14253	$12.1ns$
Master Node	3325	1923	$17.3ns$

VI. Discussion and Conclusions

This section discusses the results collected from this work, and draws some conclusions and possible lines for future developments.

a) A case for reconfigurable computing in cryptanalysis: The work presented in this paper makes a case for the role of reconfigurable computing in cryptanalytic high-performance applications. Reconfigurable technologies enable inherent new opportunities, including algorithmic and architectural exploration and static/dynamic input-dependent circuit specialization. Brand new techniques at the algorithm level, presented in Section IV, stemmed from this study, in addition to a large variety of architectural optimizations and specific design techniques.

b) Highest speed/cost ratio for SHA-1 collision search: To demonstrate the impact of the approach presented, we have developed a working FPGA-based cluster. The EOC it is able to reach is much higher than other existing software or hardware solutions. For example, the proposal in [13], based on the MariCel supercomputer featuring high-speed IBM CBE processors (each containing eight 4-slot SIMD units, called SPUs, working at 3.2GHz) is able to reach an EOC = 0.026, referred to one SPU, running a highly optimized SIMD application. Taking into account the difference in the clock frequency, that means that a single SHA-1 collision core is able to find a collision in a comparable time, precisely only around 1.19 times larger than an SPU core in the supercomputer used in [13]. Compared to the HPC MariCel supercomputer, however, building an FPGA-based cluster based on the SHA-1 collision core costs two orders of magnitude less.

Looking at hardware solutions in the literature, [6] presents a microprocessor with minimal area requirements for speeding-up the MD-4 hash function. Synthesized for a Spartan3 XC3S1000 FPGA device, their collision search unit requires around 700 slices (9% of the device resources, according to the paper), each containing two LUTs and two flip-flops, i.e. around 30% less resources than our SHA-1 collision core. On the other hand, the unit is not targeted at SHA-1 and, being based on a software-like approach, it would be considerably slower than our core. Working sequentially on 32-bit data, in fact, it would require at least 12 cycles for an EO (update of variables $A_i \dots E_i$ and W_i for two different messages) not to mention checks and control operations, with an EOC certainly (much) below $1/12 = 0.083$ if used for SHA-1.

The FPGA-based prototypical cluster presented in Section V, in conclusion, is currently the solution with the highest performance/cost ratio for SHA-1 collision search.

c) The first 72-round SHA-1 collision: At the end of our work, we used the prototypical cluster to outperform a previous result in SHA-1 cryptanalysis. In fact, we were able to find an actual collision for a 72-round version of SHA-1, beyond the limit reached in [13]. The collision, listed below, was the most advanced result towards a break of the full 80-round SHA-1 algorithm at the time of the discovery.

d) Future developments: With the support of a major FPGA manufacturer, we plan to build a large scale version of the cluster and demonstrate its potential with new results for SHA-1. At the same time, we plan to extend our approach to other hash algorithms, namely the candidates for the selection of the SHA-3 function, so as to enable an early analysis of the proposals and anticipate possible unexpected vulnerabilities.

72-round Collision

Message 1				Message 2			
Word	Value (Hex)	Word	Value (Hex)	Word	Value (Hex)	Word	Value (Hex)
0	E03BE94A	16	17E06210	0	503BE919	16	A7E06243
1	55429082	17	4228938E	1	6542908A	17	72289386
2	51F58CAD	18	8F080AE2	2	51F58CEE	18	8F080AA1
3	165504EB	19	69723D6F	3	C6550499	19	B9723D1D
4	6FA80C12	20	28629C47	4	DFA80C02	20	98629C57
5	C0C0E90B	21	371A4D30	5	30C0E969	21	C71A4D52
6	A571346E	22	8CFECD5F	6	6571342C	22	4CFECD1D
7	F541EB71	23	6A92FE7F	7	F541EB41	23	6A92FE4F
8	741493FE	24	12C368B9	8	941493BC	24	F2C368FB
9	E46596B8	25	4C5C4030	9	C46596D8	25	6C5C4050
10	257FCC2C	26	15E3BB2D	10	C57FCC6D	26	F5E3BB6C
11	68A706C2	27	DEA02DF5	11	48A70692	27	FEA02DA5
12	7A282A59	28	3D54D825	12	BA282A18	28	FD54D864
13	467204D2	29	C5FE48AA	13	A67204A0	29	25FE48D8
14	A3B36574	30	1A280A1F	14	03B36577	30	BA280A1C
15	0E1D76B2	31	D2124E30	15	CE1D76A0	31	12124E22

Hash	BD28230B0A52F51F0FEA3B7EFCBE8120EE1C4036

Fig. 4. The 72-round SHA-1 collision found by the FPGA cluster

Acknowledgment

The author would like to thank Luigi Esposito for supporting the development of the software part of the collision search application.

References

[1] T. Güneysu, T. Kasper, M. Novotný, C. Paar, and A. Rupp, "Cryptanalysis with COPACOBANA," *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1498–1513, 2008.

[2] (2011, Jan.) Rivyera S3-5000. [Online]. Available: http://www.sciengines.com/

[3] D. Suzuki, "How to maximize the potential of FPGA resources for modular exponentiation," in *Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems*, ser. LNCS, vol. 4727. Springer, 2007, pp. 272–288.

[4] G. de Meulenaer, F. Gosset, M. M. de Dormale, and J.-J. Quisquater, "Integer factorization based on elliptic curve method: Towards better exploitation of reconfigurable hardware," in *Proceedings of the 15th Annual Symposium on Field-Programmable Custom Computing Machines (FCCM) 2007*. IEEE, 2007, pp. 197–206.

[5] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D. Osvik, and B. de Weger. (2011, Jan.) MD5 considered harmful today: Creating a rogue CA certificate. [Online]. Available: http://www.win.tue.nl/hashclash/rogue-ca/

[6] T. Güneysu, C. Paar, and S. Schäge, "Efficient hash collision search strategies on special-purpose hardware," in *Research in Cryptology*, ser. LNCS, vol. 4945. Springer, 2008, pp. 39–51.

[7] F. Chabaud and A. Joux, "Differential collisions in SHA-0," in *Proceedings of CRYPTO '98*, ser. LNCS, vol. 1462. Springer, 1999, pp. 56–71.

[8] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby, "Collisions of SHA-0 and reduced SHA-1," in *Proceedings of EUROCRYPT 2005*, ser. LNCS, vol. 3494. Springer, 2005, pp. 36–57.

[9] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1," in *Proceedings of Advances in Cryptology - CRYPTO 2005*, ser. LNCS, vol. 3621. Springer, 2005, pp. 17–36.

[10] C. De Cannière and C. Rechberger, "Finding SHA-1 characteristics: General results and applications," in *Advances in Cryptology - Asiacrypt 2006*, 2006.

[11] C. De Cannière, F. Mendel, and C. Rechberger, "Collisions for 70-step SHA-1: On the full cost of collision search," in *Proceedings of SAC 2007*, ser. LNCS, vol. 4876. Springer, 2007, pp. 56–73.

[12] S. Manuel and T. Peyrin, "Collisions on SHA-0 in one hour," in *Fast Software Encryption*, ser. LNCS, vol. 5086. Springer, 2008, pp. 16–35.

[13] A. Cilardo, L. Esposito, A. Veniero, A. Mazzeo, V. Beltran, and E. Ayguadé, "A CellBE-based HPC application for the analysis of vulnerabilities in cryptographic hash functions," in *Proceedings of the 12th Conference on High Performance Computing and Communications (HPCC'10)*. IEEE, 2010, pp. 450–457.

[14] *Secure hash standard. Federal Information Processing Standard, FIPS-180-1*, NIST Std., 1995.

[15] A. Joux and T. Peyrin, "Hash functions and the (amplified) boomerang attack," in *Advances in Cryptology - CRYPTO 2007*, ser. LNCS, vol. 4622. Springer, 2007, pp. 244–263.