Efficient High-Speed WPA2 Brute Force Attacks using Scalable Low-Cost FPGA Clustering

Markus Kammerstetter[1], Markus Muellner[1], Daniel Burian[1], Christian Kudera[1], and Wolfgang Kastner[2]

[1] Secure Systems Lab Vienna, Automation Systems Group, Institute of Computer Aided Automation, Vienna University of Technology, {mk,mmuellner,dburian,ckudera} @ seclab.tuwien.ac.at,
[2] Automation Systems Group, Institute of Computer Aided Automation, Vienna University of Technology, {k} @ auto.tuwien.ac.at

Abstract. WPA2-Personal is widely used to protect Wi-Fi networks against illicit access. While attackers typically use GPUs to speed up the discovery of weak network passwords, attacking random passwords is considered to quickly become infeasible with increasing password length. Professional attackers may thus turn to commercial high-end FPGA-based cluster solutions to significantly increase the speed of those attacks. Well known manufacturers such as Elcomsoft have succeeded in creating world's fastest commercial FPGA-based WPA2 password recovery system, but since they rely on high-performance FPGAs the costs of these systems are well beyond the reach of amateurs. In this paper, we present a highly optimized low-cost FPGA cluster-based WPA-2 Personal password recovery system that can not only achieve similar performance at a cost affordable by amateurs, but in comparison our implementation would also be more than 5 times as fast on the original hardware. Since the currently fastest system is not only significantly slower but proprietary as well, we believe that we are the first to present the internals of a highly optimized and fully pipelined FPGA WPA2 password recovery system. In addition, we evaluated our approach with respect to performance and power usage and compare it to GPU-based systems.

Keywords: FPGA, WPA2, Security, Brute Force, Attacks

1 Introduction

Today's Wi-Fi networks are commonly protected with the well known WPA2 protocol defined in the IEEE 802.11 standard documents [6]. The WPA2-Personal variant is designed for smaller networks and uses a pre-shared key (i.e., a Wi-Fi password) to derive the necessary key material for authentication, encryption and integrity protection. The Wi-Fi password needs to be at least 8 characters long and the key material is mainly derived through the salted key derivation function PBKDF2 [8] in combination with the SHA1 hashing algorithm [1]

in HMAC configuration [2]. Due to the computational complexity of the key derivation function and the use of the Wi-Fi's SSID as cryptographic salt, brute force attacks are very hard to conduct in the presence of random passwords with increasing length. Incurring significant costs well outside of what amateurs can afford, professional attackers can turn to commercial high-end FPGA-based cluster solutions achieving WPA-2 password guessing speeds of 1 million guesses per second and more [10]. In this paper, we focus on the WPA2-Personal key derivation function and low-cost FPGA cluster based attacks affordable by amateurs. Especially considering second-hand FPGA boards that have been used for cryptocurrency mining, those boards are now available at low cost and can be repurposed to mount attacks on cryptographic systems. In the first part, we use a top-down approach to present WPA2-Personal security at a high level and we subsequently break it down to low-level SHA1 computations. In the second part, we use a bottom-up approach to show how these computations can be addressed in hardware with FPGAs and we present how our solution can be integrated into a scalable low-cost system to conduct WPA-2 Personal brute force attacks. We evaluate our system with respect to performance and power usage and we compare it to results we obtained from GPUs. The extended version of our paper [9] also includes a real-world case study highlighting the practical impact. Specifically, the contributions presented in this paper are as follows:

- We present a highly optimized design of a scalable and fully pipelined FPGA implementation for efficient WPA2 brute force attacks that brings the performance of today's highly expensive professional systems to the low-cost FPGA boards affordable by amateurs.
- Our implementation on Kintex-7 devices indicates that on the same hardware, our implementation is more than 5 times as fast in comparison to what is currently marketed to be world's fastest FPGA-based WPA2 password recovery system [4, 10].
- We implemented and evaluated our approach on three different low-cost FPGA architectures including an actual FPGA cluster with 36 Spartan 6 LX150T devices located on repurposed cryptocurrency mining boards.
- We evaluate our system with respect to the power consumption and performance in comparison to GPU clusters, showing that FPGAs can achieve comparable or higher performance with considerably less power and space requirements.

2 Related Work

Since WPA2 is commonly used, there are several publications and projects dealing with WPA2 security and brute force attacks in particular. For instance in [11], Visan covers typical CPU and GPU accelerated password recovery approaches with state-of-the-art tools like aircrack-ng[3] or Pyrit[4]. He considers

[3] `http://www.aircrack-ng.org` [4] `https://code.google.com/p/pyrit`

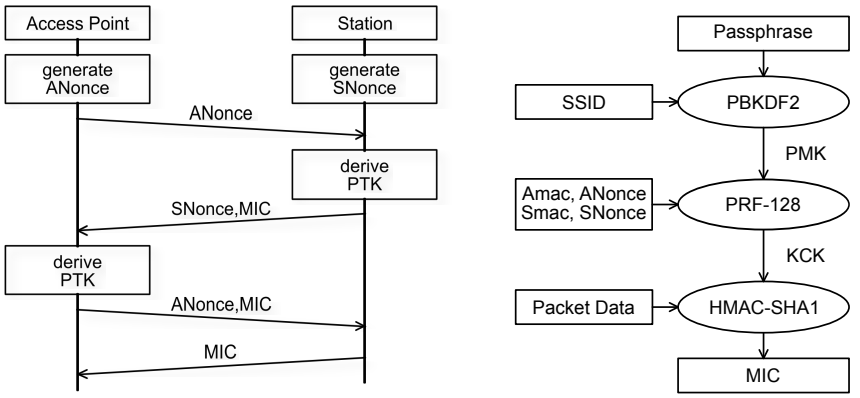
a time-memory tradeoff usable for frequent Wi-Fi SSIDs and provides a performance overview of common GPUs and GPU cluster configurations. In that respect, oclHashcat[5] and the commercial Wireless Security Auditor software[6] need to be mentioned which are both password recovery frameworks with GPU acceleration and WPA2 support. Unlike these GPU-based approaches, our system comprises of a highly optimized and scalable FPGA implementation allowing higher performance at lower costs and power consumption in comparison. In [7], Johnson et al. present an FPGA architecture for the recovery of WPA and WPA2 keys. Although WPA support is mentioned, their implementation seems to support WPA2 only which is comparable to our system. However, while our implementation features multiple fully pipelined and heavily optimized cores for maximum performance, Johnson et al. present a straight-forward and mostly sequential design leading to a significantly less performance in comparison. In [5], Güneysu et al. present the RIVYERA and COPACOBANA high-performance FPGA cluster systems for cryptanalysis. They provide details on exhaustive key search attacks for cryptographic algorithms such as DES, Hitag2 or Keeloq and have a larger cluster configuration than we had available for our tests. Yet, in contrast to our work, they do not cover WPA2 or exhaustive key search attacks on WPA2 in their work. As a result, it would be highly interesting to evaluate our FPGA implementation on their machines. Finally, Elcomsoft's commercial Distributed Password Recovery[7] software needs to be mentioned due to its support for WPA2 key recovery attacks on FPGA clusters [4, 10] and its claim to be world's fastest FPGA-based password cracking solution [3]. Although there is practically no publicly available information on the internals of their WPA2 implementation, in [10] performance data are provided. In contrast to their work, we do not only disclose our design, architecture and optimizations of our FPGA implementation, but we also claim that on the same professional FPGA hardware our implementation would be more than 5 times as fast. In comparison to the professional system, our system can achieve similar speeds on the low-cost repurposed cryptocurrently mining hardware available to amateurs.

3 WPA2-Personal Handshake and Key Derivation

In WPA2-Personal, Station and Access Point (AP) mutually authenticate against each other with the 4-way handshake depicted in Fig. 1a. To start the mutual authentication process, the AP generates a 32 byte random `ANonce` and sends it to the Station. Similarly, the Station generates a 32 byte random `SNonce` and uses both nonces as well as the password to derive the PMK (Pairwise Master Key) and the Pairwise Transient Key (PTK) with the help of the WPA2-Personal key derivation (Fig. 1b). The nonces ensure that the handshake cannot by replayed by an attacker at a later time. Afterwards, the Station sends the `SNonce` back to the AP and utilizes the `PTK` truncated to the first 128 bits (Key Confirmation

[5] `http://hashcat.net/oclhashcat`
[6] `https://www.elcomsoft.com/ewsa.html`
[7] `https://www.elcomsoft.com/edpr.html`

Key - KCK) to compute a Message Integrity Code (MIC) over the packet data. At this point, the AP can compare the received MIC with the computed one to validate that the Station is authentic and has knowledge of the password. To prove to the Station that the AP knows the password, the Station sends a message including ANonce and the corresponding MIC code. Since the Station can only compute the correct MIC code if it knows the PTK, the AP can use this information for authentication. On success, the Station completes the handshake by sending a usually empty, but signed (MIC) message back to the AP.



(a) 4-Way Handshake (b) Key Derivation

Fig. 1: WPA2-Personal Handshake and Key Derivation

During key derivation (Fig. 1b), PMK is computed from the password and the SSID as cryptographic salt through the PBKDF2 [8] key derivation function with HMAC-SHA1 at its core. The PTK and its truncated variant denoted KCK are computed through the HMAC-SHA1 based pseudo random function PRF-128. Likewise, also the computation of the MIC integrity code relies on HMAC-SHA1.

3.1 Breaking it down to SHA1 Computations

Internally, the PBKDF2 key derivation function employed in WPA2-Personal utilizes $4,096$ HMAC-SHA1 iterations to obtain 160 bit hash outputs (Fig. 2). Since the WPA2 Pairwise Master Key PMK needs to be 256 bits long, two PBKDF2 rounds are necessary. Their output is concatenated, but from the second iteration the output is truncated to 96 bits to achieve a 256 bit result. In both PBKDF2 iterations the password is used as key while the SSID of the Wi-Fi network concatenated with a 32 bit counter value serves as input. In the first iteration, the counter value is one while in the second iteration it is two. Consequently within both PBKDF2 iterations, there are $8,192$ HMAC-SHA1 iterations required to compute the PMK. In the first PBKDF2 round the xor-transformation is applied on the password and the inner pad ipad. The result is a 512 bit block serving as input to the SHA1 hash function in initial state. The output is the HMAC inner state. Since the SSID may be no longer than 32 bytes,

the hashing of the SSID and the PBKDF2 round counter can be done together with the SHA1 finalization so that only one SHA1 iteration is necessary.

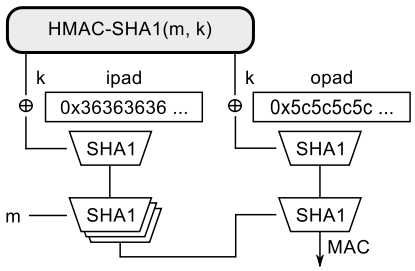


Fig. 2: PBKDF2 core with SHA1 rounds in HMAC construction

In the next step, the outer HMAC state is computed by hashing the xor of the password and the outer pad `opad`. Afterwards, the previously finalized 160 bit digest is hashed and finalized with the outer state. At this point the MAC is ready. The second PBKDF2 iteration is computed in the same way with the difference that the round counter value is set to two instead of one. Since the password does not change during PBKDF2 iterations, the inner and outer HMAC states stay the same allowing us to use cached states instead of having to compute the states again. With that optimization in mind, it is required to compute at least $2 + 4,096 * 2$ SHA1 iterations for the first PBKDF2 round and $4,096 * 2$ SHA1 iterations for the second round (i.e., 16,386 SHA1 iterations in total) to obtain the `PMK`. This computational effort, the use of the SSID as salt for key derivation and the security of the innermost SHA1 cryptographic hash function are the main reasons why WPA2-Personal key derivation is very strong against typical exhaustive key search attacks. Once the `PMK` is available, the `KCK` is derived by applying a 128 bit Pseudo Random Function (PRF). Internally, it just uses HMAC-SHA1 again with the `PMK` as key. The hashed message is made up of the string "Pairwise key expansion", a terminating zero byte, an arithmetically sorted tuple of the AP and Station addresses as well as another sorted tuple of their nonces (i.e., `ANonce` and `SNonce`) including a finalizing zero byte. The `PTK` is the resulting MAC and it is truncated to the first 128 bits to obtain the `KCK`. If the `PMK` is available, the computation of the `KCK` takes 5 SHA1 iterations as due to the length of the `PMK` the finalization of the inner HMAC state can not be combined with the hashing of the `PMK`. Whenever AP or Station would like to compute a `MIC`, they can do so by utilizing HMAC-SHA1 on the message with `KCK` as key. The result of the computation truncated to the first 128 bits is the `MIC`. The computational effort depends on the length of the message. However, considering the messages from the 4-way WPA2-Personal handshake, a total of 5 SHA1 iterations is required to compute the `MIC` since, similar to the `KCK` computation, the finalization of the inner HMAC state requires one additional iteration. A more detailed description of the key derivation is available in the extended version of our paper [9].

3.2 Attacking the 4-Way Handshake

If an attacker wants to determine the WPA2-Personal password, a 4-way WPA2-Personal handshake between a Station and AP needs to be obtained first. This can either be done passively or with the help of an active de-authentication attack where the attacker spoofs the source address of the AP and sends de-authentication frames to the Station. Since those frames are not authenticated, the Station will falsely believe that the de-authentication request came from the genuine AP and will follow the request. However at a later time, it will re-authentication and thus give the attacker the opportunity to intercept the handshake. As soon as the attacker has the handshake, passwords can be guessed offline by deriving the key material for the `PMK` and the `KCK` and computing the `MIC` for one of the observed packets in the handshake. If the observed `MIC` is the same as the computed `MIC` for a password candidate, the attacker has found the correct password for the network. However, since a WPA2-Personal password needs to have a minimum length of 8 characters and for each password candidate a total of at least $16,386 + 5 + 5 = 16,396$ SHA1 iterations are necessary to compute the corresponding `MIC` over a handshake packet, exhaustive password guessing attacks are considered to be increasingly infeasible with higher password complexity and length. In the subsequent chapters, we show that the high computational effort can be addressed with special purpose FPGA hardware so that a high number of real-world WPA2-Personal protected networks with random passwords can be broken within days.

4 FPGA Implementation

Assuming familiarity with FPGA design in general, SHA1 [12] is especially well suited for FPGA implementation due to the following reasons:

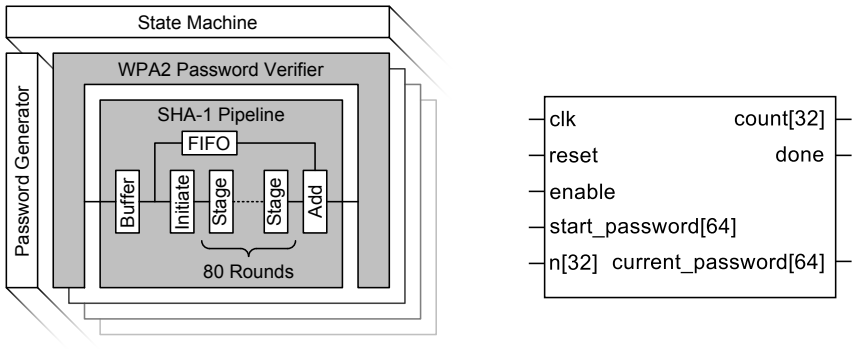
1. The algorithm has practically no memory requirements.
2. The rotate and shift operations utilized in SHA1 can be realized through FPGA interconnects with minimal time delay.
3. Algebraic logic functions (xor, and, or, not, etc.) require minimal effort and can efficiently utilize the FPGAs LUTs.

The most expensive operation are SHA1's additions due to the long carry chain between the adders. To implement the algorithm, a surrounding state machine is required to control which inputs should be supplied to the logic in different rounds. Considering that SHA1 has 80 rounds and we would like to achieve maximum performance, there are two design options: Either the SHA1 algorithm is implemented sequentially or in a fully pipelined way. The advantage of a sequential implementation is that the FPGA can be completely filled up with relatively small SHA1 cores. However, the disadvantage is that each of those cores would require its own state machine which takes up a significant amount of space. In comparison, a fully pipelined implementation does not require an internal state machine as each of the SHA1 rounds is implemented in its own

logic block. While this is a significant advantage enabling parallel processing, the drawback is that a fully pipelined implementation has much higher space and routing requirements. When using multiple cores (each containing a full pipeline), only an integer number of cores can be placed so that a significant amount of unused space might be left on the FPGA. In our implementation, we also experimented with filling up this space with sequential cores but refrained from it due to the negative effect on the overall design complexity and the lower achievable clock speeds. Due to the typically higher performance that can be achieved through pipelining and the property that we get one full SHA1 computation output per clock cycle per core, we targeted a heavily optimized and fully-pipelined approach. However, while pipelining alone has a considerable performance impact in comparison to a sequential approach, the key of obtaining maximum design performance are the optimizations. Our overall FPGA design is illustrated in Fig. 3 and has the following components: A global brute force search state machine, a shared password generator and an FPGA device specific number of brute force cores, each comprising a WPA2-Personal state machine with password verifier and a SHA1 pipeline.

Global Brute Force State Machine The task of the global brute force state machine is to constantly supply all brute force cores with new password candidates and check whether one of them found the correct password. Due to the insignificant speed impact and the advantage of lower design complexity we chose an iterative approach. Since our SHA1 pipeline comprises of 83 stages, we can concurrently test 83 passwords per brute force core. With our iterative approach, we enable the password generator and consecutively fill all brute force cores with passwords. Once all cores have been filled, the password generator is paused and we iteratively wait until all cores have completed. At that point, the password filling process is restarted. If a core finds the correct password or the password generator has reached the last password, the state machine jumps into the idle state and can accept the next working block. The penalty for this iterative approach is 83 clock cycles per core since once a brute force core has finished, we could immediately fill it with a new password. However, in comparison to the long run time of each core the impact is insignificant.

Password Generator The password generator (Fig. 3b) is realized as a fast counter. Whenever the FPGA is idle, it can accept a new working block comprising of all necessary data including the actual start password (`start_password`) and how many passwords (`n`) should be tested. Initially starting at the start password, whenever the password generator is enabled (`enable`) it will output a new password (`current_password`) and the current password number (`count`) in each clock cycle. In case no more passwords can be fed into the brute force cores, the generator can be paused at any time by disabling the `enable` input. Ultimately, it will output new passwords until `n` passwords have been reached and assert the `done` signal to indicate that all passwords within the current working block have been generated.



(a) FPGA Global State Machine (b) Password Generator

Fig. 3: FPGA Global State Machine and Password Generator Block

During the optimizations of our cryptographic cores in the design, at some point the long carry chain in the password counter became the clock speed limiting critical path. We were able to address the issue by parallelizing the counter and implementing the password carry with static multiplexers outside the sequential logic block. The sequential logic block can be seen as typical register transfer logic (RTL). With the clock signal, the old counter value is fetched from the source register, increased and finally output to the destination register. The path in between accounts for the delay. Since we need to have a carry overflow at the last valid password character (e.g., 'Z') we need a set of multiplexers that eventually reset the characters at each position of the password string. However, if this multiplexer based reset logic is within the sequential path it will also increase the time delay. By statically implementing the reset logic outside this sequential path we were able to balance the overall worst-case delays and achieved a password counter implementation that no longer accounted for the critical path in our overall design. Another password generator optimization approach we considered is utilizing multiple clock domains. The general idea is that the overall design naturally spends most of its time computing SHA1 iterations. At that time the password generator is disabled. We could thus use a less critical slower clock to generate the passwords and output them to clock synchronizing FIFO buffers directly placed next to the input of the SHA1 pipelines. As soon as a SHA1 pipeline requires a new password input, it can utilize its fast clock to drain the FIFO buffer which would in turn enable the password generator to refill the corresponding buffer at its slower clock. The advantages of this approach would be the following: First, the complexity of the password generator design can be further increased without negatively impacting the critical path. Second, the big advantage is the routing of the bus signals from the password generator to all the cores. Considering that the password generator is located at the center of the design and the passwords need to be distributed across the entire FPGA to all

brute force cores, there is a significant impact on the time-driven routing complexity and the interconnect delays that negatively impact the maximum clock speed of the overall design. By leveraging a slower clock, the passwords would be already located in the FIFO buffers next to the SHA1 pipelines of each core but they could still be read with the fast clock the SHA1 pipelines are operating on. However, since with our previously mentioned password generator optimization the critical path was no longer within the password generator domain, we did not implemented the approach. It will be covered in future work.

WPA2-Personal State Machine with Password Verifier Each brute force core has a WPA-2 Personal state machine with a password verifier. It is the most complex state machine in the overall design. Its task is to compute the `MIC` code for each password candidate with the help of the SHA1 pipeline in its center. Each computed `MIC` is compared with the `MIC` from the WPA2-Personal 4-way handshake to determine whether the password candidate was correct or not. Figure 4 shows all necessary states and state transitions.

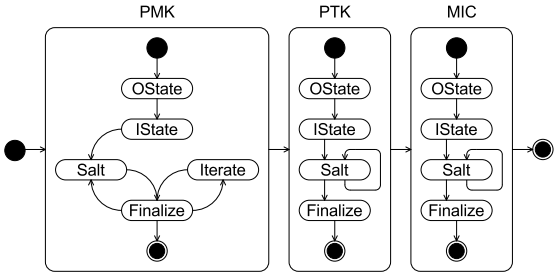


Fig. 4: WPA2-Personal FPGA States

The state machine is divided into three WPA2-Personal key derivation phases: PMK computation (1), PTK computation (2), and MIC computation (3). The computation of the PMK has the highest computation effort due to the 2 PBKDF2 rounds with $4,096$ iterations requiring $16,386$ SHA1 iterations in total. Initially, 83 password candidates and the network's `SSID` are fed into the SHA1 pipeline to compute the corresponding HMAC outer and inner states (`OState` and `IState`). Since these states do not change over the PBKDF2 iterations, the HMAC state computation needs to be done only once. In the first PBKDF2 round, the `SSID` and the PBKDF2 round counter with value 1 are used as salt. After that, there are $4,095$ more iterations in which the digest output is used as input. At that point, the second PBKDF2 round is computed by first computing the salt with an increased round counter value (2) and subsequently performing $4,095$ iterations to obtain the `PMK`.

SHA1 Pipeline In each brute force core, the SHA1 pipeline occupies a large amount of space due to the high number of pipeline stages. While SHA1 has 80 rounds and a fully pipelined implementation would thus have an equal number

of pipeline stages, we heavily optimized our pipeline to allow higher clock frequencies and consequently achieve more performance. The SHA1 pipeline is the key limiting factor of how fast our password guessing attacks can be conducted. Within the brute force cores, each of our SHA1 pipelines has 83 stages due to the optimizations we performed. Each core can thus compute 83 password candidates in parallel. The optimization approaches we applied are described in the following.

The first stage of the SHA1 pipeline is a buffer stage so that the delays of the different input logic blocks within the WPA-2 Personal state machine are not added to the pipeline's input logic and thereby do not increase the overall time delay of the critical path. The second stage denoted 'Initiate' is an optimization of the 4 required (expensive) additions in each SHA1 round. Instead of having all 4 additions in one stage, the structure of the SHA1 algorithm allows us to split up the required 4 sequential additions into two rounds with 2 additions each, thereby significantly improving the maximum clock speed. Since the SHA1 expansion steps require only a small amount of logic, another optimization is to do multiple message expansion steps in a single pipeline stage so that it is not needed in the following few stages. As a result, the source data is not accessed in each stage and shift register inference is boosted causing lower flip-flop fan-out as well as less power usage and lower area requirements. Another approach we took is the pipeline stage denoted 'Add' after the SHA1 rounds. After the last SHA1 round, the resulting digest is added either to the constant initialization vector (first iteration) or to the previous digest for subsequent iterations. Due to these expensive additions, the design performance can be improved if they are carried out in a separate pipeline stage. Instead of forwarding the initial digest through all stages to the final addition stage, we leverage a FIFO-based delay line utilizing the FPGAs Block-RAM resources. This avoids excessive interconnect routing through all stages and thus makes the design smaller, reduces the number of critical paths and allows us to achieve higher clock frequencies more easily.

Additional FPGA Design Optimizations In the WPA2-Personal state machine, we directly use the output from the password generator and compute the HMAC `OState` state first. At the same time, we store the password candidates in a Block-RAM buffer for later `IState` computation. After that, we no longer work with the passwords but use password offsets instead. The result is a lower design density as no more additional interconnects are required for the password in later stages. A similar approach is used to avoid excessive interconnects and design density. Instead of having large buses, we either use Block-RAMs directly or form RAM-based delay lines to keep the `IState` and `OState` states as well as the computed `PMKs` and `PTKs` in memory. Instead of one large WPA2-Personal state multiplexer directly controlling all SHA1 pipeline inputs and outputs, we make use of several smaller and less complex multiplexers. Once again, this reduces overall design complexity and allows us to achieve higher clock speeds more easily. The top-level design needs to communicate with the outside world. Each time a new working block is added, all necessary Wi-Fi

and WPA2-Personal data needs to be transferred and subsequently forwarded to all brute force cores. The result is a very broad bus spreading all over the FPGA design and causing severe design congestion. Since in our design only the password candidates and the `SSID` are required early within the WPA2-Personal state machine, we transfer the rest of the data over a small 16 bit bus leveraging inferred shift registers. This significantly reduces the complexity of the interconnects between the shared global state machine and the brute force cores across the FPGA. To lower the amount of input and output data exchanged with the outside world, we use a minimized Wi-Fi and WPA2-Personal data set that only includes the variable data fields from the captured handshake. All other data is not only fixed within the FPGA, but also kept locally in the cores. In addition, the FPGA does not output the correct password, but a numeric offset from the start password instead. To avoid design congestion and to push the design to the highest clock speed possible, we make use of custom parameters within the Xilinx design tools for synthesis, mapping and routing such as the minimum inferred shift register size, register balancing or the number of cost tables. In addition, we use floor planning to support the mapper, placer and router in achieving higher clock rates. Floor planning is important to place critical components requiring a fast interconnect in between next to each other. In general, we were able to obtain the highest speed improvements by utilizing a star like topography: The password generator is distributed over the very center of the FPGA and the brute force cores are surrounding it. In addition we also used floor planning to avoid the placement of time critical components in FPGA areas that are hard to reach through interconnects. Consequently, we carefully placed critical components like the SHA1 pipelines in a way that those regions do not negatively impact the routing delay. In our FPGA implementations, we use a slow clock for communication with the outside world and a fast clock for computation at the same time. In our Spartan-6 implementation, the speed of the fast clock can be adjusted dynamically during runtime by programming the clock multiplier. In contrast, our Artix-7 implementation includes an automatic clock scaling mechanism to adjust the fast clock frequency with the device core temperature. Both approaches allow the FPGA design to run at high speeds without the danger of overheating.

4.1 Overall System Design

Targeting FPGAs well in the range of amateurs, we implemented and practically evaluated our system on Xilinx Spartan-6 and Xilinx Artix-7 FPGAs. The Spartan-6 FPGAs are located on low-cost repurposed cryptocurrency mining boards. For comparison purposes, we created a full implementation for the more expensive Xilinx Kintex-7 XC7K410T FPGAs utilized in Elcomsoft's system as well, but could not practically test it since we did not have one of these FPGAs at hand. The overall system design for the Spartan-6 FPGAs is visible in Fig. 5

and based on ZTEX FPGA boards[8]. The Artix-7 design is similar but has only one XC7A200T FPGA on the board.

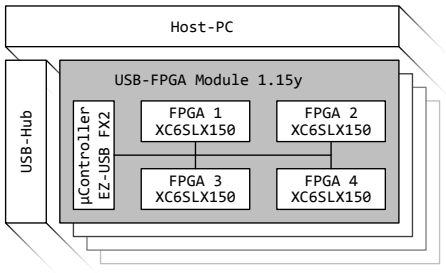


Fig. 5: System Overview (Spartan 6 System)

The system comprises of a PC with a host software and several FPGA boards connected via the USB 2.0 high-speed interface. Each FPGA board has a fast EZ-USB FX2 micro-controller with custom firmware to interface with the FPGAs. Our custom host software utilizes the ZTEX SDK to allow easy communication with the micro-controller and the FPGAs. The host software accepts a configuration file that includes all necessary Wi-Fi and WPA-2 Personal handshake data. At startup, it enumerates all connected FPGA boards, uploads the micro-controller firmware if necessary and configures the FPGAs with our bit stream. The software makes use of several threads. Apart from the main program, there is a thread to generate password working blocks for the FPGAs and additional threads for each FPGA board. The password working blocks are kept in a pool with constant size. The device threads can supply working blocks to FPGAs and mark them as being processed. If an FPGA has finished a block, it is removed from the pool and the generator automatically creates a new working block. If for some reason an FPGA fails, the block sent to the FPGA is still in the pool and just needs to be unmarked so that the next free FPGA can process it instead. The micro-controller firmware is responsible for USB communication with the host and communication with the FPGAs.

5 Evaluation

We performed multiple evaluations with regard to our design performance, the power usage and performance in comparison to GPUs. We evaluated the performance and the power usage of our design on multiple FPGAs and FPGA boards. The first FPGA we targeted was a Spartan-6 XC6SLX150T-3 device. Four of these FPGAs can be found on the Ztex 1.15y board visible on the left of Figure 6. The second FPGA we used for our evaluation was an Artix-7 XC7A200T-2 device on the Ztex 2.16 board visible on the right of the picture. For both FPGAs,

[8] `http://www.ztex.de`

we created an optimized implementation and a configuration bit stream that can be uploaded to the device. The main difference between the bit streams is the FPGA type, the maximum clock frequency and most importantly the number of brute force cores we were able to fit onto the device.

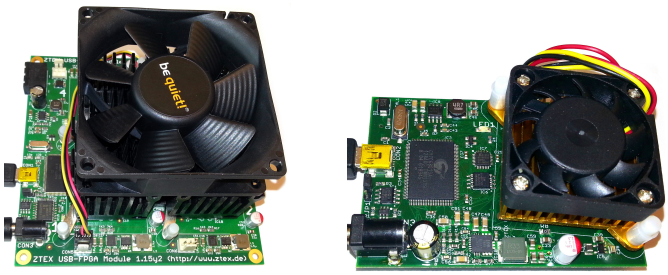


Fig. 6: Ztex 1.15y Board (left), Ztex 2.16 Board (right)

To evaluate the performance and the power requirements, we used the obtained timing and power reports by utilizing the Xilinx timing and power analysis tools. In addition to these results, we conducted practical measurements on the FPGA boards. At first, we measured the idle wattage of each unconfigured board at the power supply to determine the idle power usage. In the next step, we used a generated WPA2-Personal handshake with our software to mount a brute force attack on each of our FPGA boards. We used large password working packages resulting in a 30 seconds runtime per FPGA to avoid I/O bottlenecks. By measuring the wattage again during operation, we were able to determine the overall power consumption. To reduce the influence of the power consumption caused by losses in the power supplies or components other than the FPGA, we obtained the power consumption of our FPGA implementation through the difference between the overall idle consumption and the consumption during operation. In Section 5.1, we use the same method to determine the power consumption of GPUs to get results that can be compared to the FPGA power consumption. To obtain brute force performance measurements as well, we let each system run for at least 1 hour and computed the performance by measuring the number of password guesses during that time. The result is the average number of password guesses per second. In addition to these evaluations, we executed the implementation on our FPGA cluster with 36 Spartan-6 XC6SLX150T FPGAs located on 9 Ztex 1.15y FPGA boards. The cluster setup allowed us to perform measurements on a larger setup and to determine how well our design scales with an increasing number of FPGAs. Using the power and performance measuring methodology from above, we obtained measurement results for the cluster as well. To allow comparison with the commercial Elcomsoft password recovery system [4, 10], we created an implementation and a configuration bitstream for the more expensive Kintex-7 XC7K410T-3 devices as well. However, since we did not have a board with this type of Kintex-7 FPGA, we can provide the Xilinx development tool's timing and power analysis results only.

5.1 GPU Comparison

To measure performance and power requirements of GPUs, we utilized cudaHashcat[9] v1.36 to mount brute force attacks on the same WPA2-Personal handshake we used previously to test our FPGA implementations. We executed the tool on machines with different Nvidia GPUs (GeForce GTX 750 Ti, GeForce GTX770 Windforce OC, GRID K520) and measured the performance in passwords per second as well as the power consumption. We applied the same power measurement methodology as during our FPGA evaluation. For the Amazon EC2 GPU cloud machines with GRID K520 GPUs, we were unable to obtain power measurements. The specific machine configurations and results are described in detail in Section 6.

6 Results and Discussion

System	FPGAs	Type	Cost	Cores	Tool W	Tool MHz	Meas. W	Act. MHz	calc pwd/s	pwd/s	pwd/s W
Ztex 1.15y	1	XC6SLX150T-3	175	2	4.281	187	6.99*	180	21,956	21,871	3,128*
Ztex 1.15y	4	XC6SLX150T-3	700	8	17.124	187	27.96	180	87,826	87,461	3,128
9x Ztex 1.15y	36	XC6SLX150T-3	2,400	72	154.116	187	254	180	790,436	741,200	2,918
Ztex 2.16	1	XC7A200T-2	213	8	10.458	180	11.04	180	87,826	87,737	7,947
N/A	1	XC7K410T-3	2,248	16	25.634	216	N/A	N/A	210,783	N/A	N/A
N/A	48	XC7K410T-3	107,904	768	1,230.432	216	N/A	N/A	10,117,584	N/A	N/A

Table 1: Performance and Power Results of our Implementations for different FPGA Devices and Systems/Boards

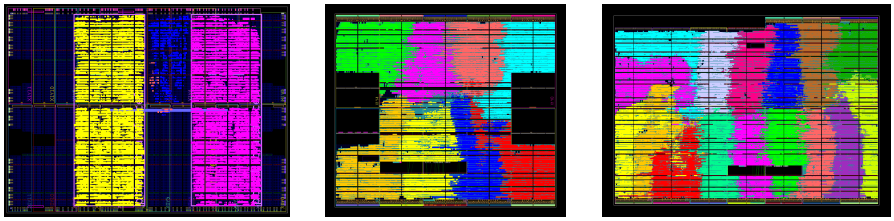
The results for our FPGA performance and power evaluation are visible in Table 1. In the `System` and `FPGA` column the table shows on which systems we conducted our tests and how many FPGAs there are on the corresponding board and/or in the overall system. The FPGA device types are visible in the `Type` column whereat the name before the hyphen is the Xilinx device name and the number after the hyphen indicates the device speed grade (the higher the better). The `Cost` column provides an approximate cost estimate per FPGA in US$ we obtained by looking up the devices at common Xilinx distributors such as Digi-key[10]. However while the cost for 9 new Ztex 1.15y would be appoximately $6,300$ US$, we considered our 9 second-hand Ztex 1.15y boards previously used for cryptocurrency mining instead. We were able to obtain these boards for $2,400$ US$ which we believe is what amateurs could do as well, depending on how much boards they would like to acquire and how much they are willing to spend. The `Cores` column shows how many cores we were able to fit onto the device to achieve maximum performance. While more cores per device generally increase the performance, it can also cause the maximum clocking speed to drop significantly due to mapping, placement and routing issues. The table presents the implementations allowing us to achieve the maximum performance per device. The `Tool W` and `Tool MHz` columns present the design

[9] `http://hashcat.net/oclhashcat` [10] `http://www.digikey.com`

tool's power and timing analysis results. For the Spartan-6 FPGAs, we used the Xilinx ISE Suite 14.7 whereas for the newer 7-series devices Artix-7 and Kintex-7, we used Vivado Design Suite 2015.1. In general, it appeared that the newer Vivado tools produced better results, but since it doesn't support older model 6-series devices, we were unable to use it for our Spartan-6 implementations. The `Meas. W` and `Act. MHz` columns present the results for the power measurements we conducted on the FPGA boards/systems and the actual clock speed we used to run the devices. The `calc pwd/s` and `pwd/s` columns provide the WPA2-Personal performance in passwords per second whereas the first one indicates the calculated and theoretic maximum performance of our implementation whereas the latter one shows the actual measured average performance per board and/or system. In the last column `pwd/s W`, we use our actual power and performance measurements to determine how much brute force speed can be achieved per Watt which is especially important when scaling up our implementation to larger FPGA cluster systems. In the following, we discuss the results of our implementations on a per-device basis.

Spartan-6 Results We used the Xilinx Spartan-6 XC6SLX150T-3 FPGA as the target for our first implementation due to the availability of a high-performance FPGA cluster with 36 of these devices at our lab. The implementation on the Spartan-6 turned out to be especially challenging for multiple reasons. We had to deal with long design tool runs (3 hours of more) each time we made modifications to the design. Since the effects of many of our optimizations could not be tested through simulations alone, the duration of the design tool runs significantly slowed down the development. In addition, the internal switch boxes and types of slices in the Spartan-6 architecture are not well suited for more complex and larger implementations in comparison to newer 7-series devices. An important factor to achieve routable designs was our use of FPGA floor planning. Our optimized 2 core implementation visible in Fig. 7a is able to run at up to 187 MHz leading to the highest performance we were able to achieve on this device. The picture shows the ready-to-upload placed and routed design. On the left and right the 2 brute force cores are clearly visible. In between the password generator and the global state machine are located. Although the dark areas indicate that there would be sufficient space for an additional core, our experiments showed that this would lead to lower performance due design congestion. The first 3 rows in Table 1 present the results we obtained through this implementation. Due to cooling requirements, we ran the design with a reduced clock speed of 180 MHz. Our measurements indicate that in this configuration, our implementation requires a total of 27.96W for all 4 FPGAs on the Ztex 1.15y board. The power measurements per Spartan-6 FPGA are marked with an asterisk to indicate that we were unable to measure them directly, but rather derived the measurement results from our power measurements for the entire Ztex 1.15y board with its 4 FPGAs. Our results show that our approach scales well and can be easily run in a cluster configuration producing a performance of $790,436$ password guesses per second on our cluster. The difference between

the calculated maximum performance and the measured performance is mainly due to the I/O times between the PC, the microcontroller and the FPGAs. In addition, our Spartan-6 implementation includes a dynamic frequency scaling mechanism slowing down the FPGAs in case of device temperatures getting too high. With better cooling inside the cluster, we believe that the gap between the theoretic performance and the measured performance could be made smaller.



(a) XC6SLX150T (b) XC7A200T (c) XC7K410T

Fig. 7: Placed and Routed FPGA Designs

Artix-7 Results Starting from our already highly optimized Spartan-6 design we ported our implementation to the newer 7-series Artix-7 XC7A200T-2 FPGA. Since device internals such as the clocks or PLLs are different from the Spartan-6 architecture, we had to adapt our implementation accordingly. The ability to read the device's core temperature from within the FPGA implementation was especially interesting. It allowed us to implement frequency scaling mechanisms directly on the FPGA not only preventing possible damage due to overheating, but also ensuring that each device always runs at the maximum performance possible. Our ready-to-upload placed and routed design is visible in Fig. 7b. Through floorplaning all of the cores have a small path to the center where the small block with the global state machine and the password generator are located. The implementation can be run at up to 180 MHz to achieve a theoretic maximum of $87,826$ password guesses per second. With a measured performance of $87,737$ password guesses per second, our results show that a single XC7A200T-2 device achieves not only more performance than 4 of the older model Spartan-6 XC6SLX150T-3 FPGAs altogether, but it also requires just 11.04 Watt during operation.

Kintex-7 Results In contrast to the low-cost Artix-7 FPGAs, Kintex-7 FPGAs are larger and allow higher performance but are also significantly more expensive. Although we didn't have any of those FPGAs at hand, we created an implementation for the Kintex-7 XC7K410T FPGA for two reasons. First, Elcomsoft's marketed to be world's fastest FPGA-based WPA2 password recovery system relies on these FPGAs just the same and even provides performance figures for it [10]. Our targeting of the same FPGAs thus allows direct performance comparison between their implementation and ours. Their document indicates that on the PicoComputing SC5/M505-48 cluster with 48 XC7K410T FPGAs their implementation is able to produce $1,988,360$ passwords guesses per second [10]. Assuming that their implementation targets WPA2 employing

SHA1 instead of WPA1 employing the much less complex MD5 algorithm, our implementation could achieve up to 10,117,584 passwords per second on the same hardware and would thus be more than 5 times as fast. Second, we wanted to obtain performance data for larger FPGAs as well. Although expensive, we believe that Kintex-7 FPGAs are well in the price range for professional attackers allowing them to achieve significantly more brute force attack performance per FPGA in comparison to low-cost FPGAs such as the Artix-7. Our ready-to-upload placed and routed design is visible in Fig. 7c. It comprises 16 cores running at up to 216 MHz. Similar to our Artix-7 implementation, the password generator and the global state machine are located in the center. At the same time, the image suggests that with an increasing number of cores, the centralized state machine and password generator becomes a bottleneck due to the long bus interconnects reaching to the outside cores. We believe that this problem could be easily addressed by including FIFOs for the password candidates in each of the brute force cores.

6.1 GPU Results and Comparison

The results of our GPU evaluation (Section 5.1) are visible in Table 2. We performed the performance measurements by running cudaHashcat v1.36 on different systems and measuring the power consumption as the difference between idle and busy WPA2 computations to get results independent from other components in the system. The table shows the different GPU configurations (`System`) we used for our tests. The `pwd/s` column shows the performance in passwords per second and the `W` column indicates the power consumed by the GPU during runtime in Watt. The performance per Watt is visible in the `pwd/s W` column. In addition to running GPU measurements on our own machines, we also conducted measurements on dedicated Amazon Elastic Cloud (EC2) GPU machines as well. While we could measure the performance on the machines just the same, we were unable to obtain power measurements. Although using a high number of GPU cloud machines appears promising to achieve high brute force attack performance, the limiting factor is the cost. Although our combined experiments on the dedicated Amazon EC2 machines took no longer than an hour, the costs we accumulated for our tests were already US$ 14.92. Since realistic brute force attacks might take considerably longer, the costs for an attacker would be far lower for acquiring a powerful GPU system instead of using the Amazon EC2 GPU nodes. In comparison to the results we obtained from our FPGA implementation, it is visible that GPUs can achieve the performance of a state-of-the-art low-cost FPGA (i.e., Artix-7), but their power consumption and performance per Watt is more than 10 times as high. At the same time, the performance achievable with a single larger FPGA such as the Kintex-7 XC7K410T is no longer in the range of GPUs. Considering high-speed attacks with clusters, we believe that the scalability for FPGA-based attacks is better due to the small size of FPGAs, their lower power consumption and the high performance they can produce.

System	pwd/s	W	pwd/s W
GeForce GTX750 Ti	52,446	106	495
GeForce GTX770 OC	62,420	184	339
Amazon EC2 - GRID K520	30,370	N/A	N/A
Amazon EC2 - GRID K520 x4	109,073	N/A	N/A

Table 2: Performance and Power Results on GPUs

7 Conclusion and Future Work

In this paper, we demonstrated that WPA2 passwords can be attacked at high speed rates not only by expensive professional FPGA cluster solutions but similar speeds can be achieved by amateurs on a low budget as well, especially when considering second hand FPGA boards previously used for cryptocurrently mining. We specifically targeted low-cost FPGA devices, conducted implementations on 3 different FPGA architectures and evaluated our results with regard to performance and power. Our GPU evaluation suggests that FPGAs can not only achieve higher speeds at significantly less power, but they can also be used to easily create small and afforable FPGA clusters in the reach of amateurs. A case study highlighting the practical impact of our attacks as well as a more detailed description of our approach is available in the extended version of our paper [9]. However, we believe that besides the speedup we achieved it is more important to consider that the WPA2-Personal brute force performance achievable on professional systems is now becoming feasible on the low-cost systems amateurs can afford as well. As counter measure, users need to increase the length of their passwords, the password should be random and it should utilize a large character set to increase password entropy. In future work, we are looking forward to evaluate the security of other cryptographic systems as well. In that regard, we plan to design and implement a powerful low-cost FPGA cluster similar to COPACOBANA [5] but with low-cost 7-series devices instead.

Acknowledgments

The research was funded by the Austrian Research Funding Agency's (FFG) KIRAS security research program through the $(SG)^2$ project under national FFG grant number 836276, the AnyPLACE project under EU H2020 grant number 646580, and the IT security consulting company Trustworks KG who also provided the FPGA boards and the cluster.

Bibliography

[1] D. Eastlake 3rd and P. Jones: US Secure Hash Algorithm 1 (SHA1). RFC 3174 (Informational) (Sep 2001), updated by RFCs 4634, 6234

[2] D. Eastlake 3rd and T. Hansen: US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). RFC 6234 (Informational) (May 2011)

[3] Elcomsoft: ElcomSoft and Pico Computing Demonstrate World's Fastest Password Cracking Solution. https://www.elcomsoft.com/PR/Pico_120717_en.pdf, [Online; accessed 13-Nov-2015]

[4] Elcomsoft Blog: Accelerating Password Recovery: the Addition of FPGA. http://blog.elcomsoft.com/2012/07/accelerating-password-recovery-the-addition-of-fpga (2012), [Online; accessed 13-Nov-2015]

[5] Güneysu, T., Kasper, T., Novotný, M., Paar, C., Wienbrandt, L., Zimmermann, R.: High-Performance Cryptanalysis on RIVYERA and COPACOBANA Computing Systems. In: Vanderbauwhede, W., Benkrid, K. (eds.) High-Performance Computing Using FPGAs, pp. 335–366. Springer New York (2013), http://dx.doi.org/10.1007/978-1-4614-1791-0_11

[6] IEEE-Inst.: 802.11-2012 - IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Tech. Rep. IEEE Std 802.11™-2012, IEEE-Inst (2012), http://ieeexplore.ieee.org/servlet/opac?punumber=6178209

[7] Johnson, T., Roggow, D., Jones, P., Zambreno, J.: An fpga architecture for the recovery of wpa/wpa2 keys. Journal of Circuits, Systems, and Computers (JCSC) 24(7) (2015)

[8] Kaliski, B.: PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898 (Informational) (Sep 2000), http://www.ietf.org/rfc/rfc2898.txt

[9] Kammerstetter, M., Muellner, M., Burian, D., Kudera, C., Kastner, W.: Efficient high-speed wpa2 brute force attacks using scalable low-cost fpga clustering (extended version). http://arxiv.org/pdf/1605.07819v1.pdf, [Online; accessed 25-May-2016]

[10] PicoComputing Inc.: SC5-4U Overview. http://picocomputing.com/brochures/SC5-4U.pdf, [Online; accessed 13-Nov-2015]

[11] Sorin Visan: WPA/WPA2 Password Security Testing using Graphics Processing Units. Journal of Mobile, Embedded and Distributed Systems 5(4) (2013)

[12] U.S.Department of Commerce National Institute of Standards and Technology: FIPS PUB 180-2, Secure Hash Standard (SHS) (2002), U.S.Department of Commerce/National Institute of Standards and Technology