Detection Of Computational Intensive Reversible Covert Channels Based On Packet Runtime

Tobias Schmidbauer[1]* and Steffen Wendzel[1,2]

[1]Fernuniversität in Hagen, 58084 Hagen, Nordrhein-Westfalen, Germany
tobias.schmidbauer@studium.fernuni-hagen.de, steffen.wendzel@fernuni-hagen.de

[2]Worms University of Applied Sciences, 67549 Worms, Rheinland-Pfalz, Germany
wendzel@hs-worms.de

Received: November 24, 2021; Accepted: February 10, 2022; Published: March 31, 2022

Abstract

In current research, reversible network-level covert channels are receiving more and more attention. The restoration of the original data leaves little evidence for detection, especially if the implementation is plausibly deniable. Recently, such a channel based on one-time password hash chains has been published. The covert channel uses repeated computational intensive operations to restore a modified hash and to extract covert information transferred within. In this paper, we present an approach that observes the influence of repeated MD5, SHA2-384, SHA3-256 and SHA3-512 hash-operations on packet runtimes. Besides these hash algorithms, we also investigate whether the alphabet that the Covert Sender and the Covert Receiver agreed upon, has an influence on our detection approach. For each algorithm, we carry out three experiments with different alphabets: one without a covert channel, one with a covert channel altering all hashes, and finally, one with a covert channel altering every second hash. We further repeat each experiment ten times and define a threshold for packet runtimes without modified hashes. Also, we investigate the detectability of computational intensive reversible covert channels for all our scenarios and evaluate the detection rate depending on the number of observed packets. In addition, we describe countermeasures and limitations of our detection method and, finally, discuss application scenarios for existing network environments.

Keywords: network steganography, anomaly detection, reversible steganography, computational intensive

1 Introduction

Covert channels (further on also called CC in this paper) have experienced a dynamic development in the last few years with many new approaches that enable the transfer of covert information. One of these developments are reversible CCs, which allow the restoration of original information that were modified by the Covert Sender (CS). Besides other implementations of such channels, the publication [1] describes how covert information can be encoded in hashes of **one-time password** (OTP) hash chains. For this approach, the CS modifies a bit of the chain, while the CR recalculates the original hash and extracts the covert information according to an alphabet on which both have agreed in advance. In addition to the reversibility, the covert communication is also plausibly deniable because it randomly flips bits of a

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, 13(1):137-166, Mar. 2022
DOI: *10.22667/JOWUA.2022.03.31.137*

*Corresponding author: Faculty of Mathematics and Computer Science, Fernuniversität in Hagen, 58084 Hagen, Nordrhein-Westfalen, Germany, Tel: +49-6241-509-213

cryptographic hash, where each bit is equally likely to occur. Due to the plausible deniability and the reversibility, existing detection approaches cannot be applied trivially.

In other research areas that are focusing on anomaly detection, such as malware detection, various methods have been developed to discern suspicious behavior. Such detection approaches base, for example, on resource consumption to detect additional malicious activities, such as the power consumption of malware on mobile devices [2]. Resource consumption-based detection may also be applied to characteristics of CCs, especially for computational intensive reversible CCs. Instead of utilizing the power consumption, one might also focus on the time consumption of repeated hash operations that leads to elongated packet runtimes.

In this paper, we analyze the network-side detectability of a recently proposed CC that exploits OTPs in hash chains [1]. We show that the channel is, in fact, detectable, but only under specific conditions. Our approach is based upon the elongated runtime of network packets due to the repeated calling of computationally intensive operations. We further demonstrate that CC traffic can be detected after few packets, as far as certain conditions are fulfilled. The key contributions of this paper are:

- An indicator for the presence of a reversible computational intensive network CC based upon the runtime of packets and their acknowledgment.
- A method to detect computational intensive CCs without controlling a participant of the CC.
- Provision of results that indicate that the utilized alphabet influences the performance of a computational intensive CC and its detection.
- Evaluation of the detection rates depending on the number of observer packets.
- Provision of application scenarios and an analysis of (potential) countermeasures for our detection approach.

This paper extends the already published version [3] with new results. The additional key contributions are the investigation of two more hash-algorithms, SHA2-384 and SHA3-hashes with a length of 256bit. Further, we present additional statistical metrics dependent on the number of observed packets and extended our work by a description of potential warden placements and an application scenario of our approach for existing network environments.

The remainder of this paper is structured as follows. In Section 2, we describe fundamentals and discuss related work. Section 3 presents our detection approach and the linked experimental setup. The evaluation of our work can be found in Section 4, including statistical significance tests, runtime evaluation and detection in dependence of the number of observed packets, as well as countermeasures and limitations. In Section 5, we will discuss potential warden placements in existing network environments. Finally, we conclude our work in Section 6.

2 Fundamentals & Related Work

In the following subsections, we first explain the fundamentals of hashes and the concept of hash chains. Further, we describe their exploitation for reversible and deniable CCs. Finally, we cover related work.

2.1 Fundamentals

In this subsection, we explain the basics of cryptographic hash functions and hash chains. We also introduce OTPs and CCs. Finally, we explain how an OTP-based CC can be implemented according to [1].

2.1.1 Hashes, Hash Chains and One-Time Passwords

Hash functions map an infinite codomain to a finite set of hash values. For instance, a (theoretically) infinite input-string S is compressed by the hash function f to an output-string H of fixed length. This process is described by $f(S) \rightarrow H$ [4]. For a **cryptographic** hash operation, a so-called one way hash function is necessary as it is defined in [5]. It additionally satisfies the following three conditions:

- It is computationally not feasible for a given hash H that was computed by the function f, to determine the input that was originally given to f. This condition is also called **preimage resistance**.
- It is impossible to find $S' \neq S$ such that f(S') = f(S), also called **second preimage resistance**.
- It is impossible to find two input-strings that compute the same hash. This is also called **collision resistance**.

There exist several hash algorithms that satisfy those requirements. The three relevant algorithms considered in this paper are the *Message Digest 5 (MD5)* algorithm [6], the *Secure Hash Algorithm 2 (SHA2)* with the hash-length of 384 bit [7] and the *Secure Hash Algorithm 3 (SHA3)* algorithm with hash-lengths of 256 bit and 512 bit [8]. The SHA2 and SHA3 algorithms will from now on be referred to as SHA2-384, SHA3-256 and SHA3-512, according to the utilized hash-length. MD5 is considered outdated nowadays, for example, due to the potential violation of collision resistance [9]. SHA2-384 is still considered as secure and has the advantage over SHA-256 and SHA-512 of not being susceptible to length extension attacks. SHA3 on the other hand can be considered, besides a few other algorithms, the nowadays distinguished standard and implements the cryptographic hash function KECCAK [10] — regardless of the utilized bit length. The resource consumption of each cryptographic hash algorithm differs, depending on its implementation. Of the algorithms in this paper, MD5 consumes significantly less computing time than SHA2-384, SHA3-256 and SHA3-512.

Hash functions can be repeatedly applied. E.g., for the seed s=x_0 and $x_{i+1} = f(x_{i-1})$ with i= (0,1,2...n), there exists a sequence of computed hashes depending on the previous hash. This sequence is also called a **hash chain**. Due to the utilization of cryptographic hash functions, the previous hash value cannot be computed from a given hash. This is ensured due to the preimage resistance of cryptographic hash functions. An **OTP** is a password authentication mechanism where each password is only used once for each authenticated session. This prevents the guessing of passwords, as each password is as statistically probable as each other possible password. Such a hash chain based authentication was described in [11]. To accomplish this, A has to generate a hash chain X = $\{x_0, x_1, ..., x_n\}$ from a secret password given for x_0. To authenticate A, the authenticator B has to know the hash x_n. x_n has to be kept secret, e.g., the channel to submit it has to be secure. To perform an authentication, A now has to send the hash x_{n-1} to B. B now tests if f(x_{n-1})=x_n. If this test is passed, B authenticates A and replaces the hash x_n with x_{n-1}. For the next authentication, A has to send x_{n-2} and so on, so each password can only be used once. Since the key generation algorithm and the registration process are known, but the key itself is kept secret, an authentication based upon [11] implements Kerckhoff's principle.

2.1.2 Covert Channels

CCs are a concept to exchange covert information between two or more participants. To perform this type of steganography, a so-called overt channel is enhanced or modified with additional covert information by a CS. This addition can be extracted by one or more Covert Receivers (CR). Besides other possible fields of application, this type of hidden communication can be implemented within computer networks (for example described in [12]) or within inter-process communication (for instance described in [13]). A CC is called **reversible** if the original message can be restored to an exact image of itself after the additional

covert information has been extracted [14]. Mazurczyk et al. showed in [15] that covert network channels can either be *fully reversible* if the original information can be restored or *quasi-reversible* if only parts of the original information can be restored. Further, the CC is called **plausibly deniable** if the information transferred from the Overt Sender (OS) to Overt Receivers (OR) is modified by the CC participants in such a way that it also could have been created in the same way without covert communication [16].

Covert Channels based on OTP Hash Chains A reversible and plausibly deniable CC based on OTP Hash Chains has been delineated in [1]. Therein, the authors described the setup for the authentication-information flow as follows: OS $\rightarrow$ CS $\rightarrow$ CR $\rightarrow$ OR. However, as described within [1], this CC is not restricted to this communication scenario. The CS might be identical to the OS as well as the CR might identical to the OR and even for inter-process CCs the channel can be implemented. For an OTP authentication based upon a hash chain, the first hash (x_n) (further also called *expected hash*) has to be submitted to the OR. The CS embeds the hidden information within the submitted subsequent hash ($x_n - 1$) by flipping one specific bit. This creates a new modified hash ($x_{modified}$). The position of the modified bit defines the submitted information, i.e., CS and CR have to agree on an alphabet in advance to exchange information. The CR extracts the information by successively flipping each bit of $x_{modified}$ singularly after another, performing the hash function repeatedly and creating a hash $x_{current_char}$. Further, the CR checks for each bit modified if $f(x_{current_char}) \rightarrow x_n$. If the check is passed, the original information has been restored and the covert information has been extracted. This can be accomplished under the condition that CR knows the expected hash x_n that initially was sent to the OR. Subsequently, the CR forwards the reconstructed original hash (x_{n-1}), rendering it a fully-reversible CC.

The authors described some possible detection methods. As the CR has to perform several hash operations, they stated that this may be monitored by an observer logging the runtime of network packets, which is investigated in this paper.

2.2 Related Work

CCs have been investigated intensively in the last decades and can be implemented in various communication scenarios, such as network environments [17, 12, 18, 19] or inter-process communications [13, 20]. The variations of CCs have been rapidly growing in the last few years [21, 22], and lead to the urge for an appropriate categorization, which has been satisfied in [18]. Network-based *reversible* CCs have been described in [15] and further have for instance been investigated in [23]. The related domain of invertible watermarking has been discussed in [24, 25].

Abad [26] described a CC based upon altering the non-cryptographic hash of IP packet checksums. To the best of our knowledge, Keller et al. have described the first and, until the submission of this paper, only reversible and plausibly deniable CC based upon OTP Hash Chains in [1].

In addition to CC hiding methods, also approaches for CC detection have been researched in the last decades and classical examples can for instance be found in [27, 19, 28].

The methods from Cabuk et al. [29] investigate the inter-arrival times of network packets and were originally designed to detect timing covert channels by applying three approaches: compressibility, ε-similarity and regularity. To detect CCs through the compressibility, the number of zeros behind the comma of packet inter-arrival times are encoded as ASCII characters, where *A* represents no zeros, *B* represents one zero and so forth. The ASCII character is then concatenated with the rounded inter-arrival time. A string consisting of usually 2,000 of such concatanated sub-strings is then compressed with *gzip*. The compression factor indicates, whether specific characters occur more often (such as regular manipulated inter-arrival times of covert channels). For the ε-simmilarity, the flows first have to be sorted in ascending order and the pairwise timing difference between the current and previous sorted value is calculated afterwards. Finally, an ε-value has to be defined and timing differences below this value are

utilized as an indicator for a CC. The last method, regularity, splits the inter-arrival times ins so called windows, each consisting 2,000 flows and calculates the standard deviation of each window, which is compared to the standard deviation of other windows. In [30], the authors proposed to slightly vary these three methods of Cabuk et al. and applied them to CCs that utilize the modulation of packet sizes, which would not have been detected by the original methods. Additional threshold-based detection heuristics have been investigated in [31, 32] – just to mention a few.

Recently, a tool called WHISPER that can be used for runtime-detection of cache side-channel attacks has been described in [33]. Therein, the authors use the runtime of selected hardware operations, like FLUSH, REALOAD, PRIME and PROBE. The runtimes of each operation is first extracted and then analyzed by various machine learning algorithms to detect potential side channels. In one of the latest detection publications, the possibility of protocol-independent detection methods were researched in [34] and the utilization of machine learning for CC detection has, for example, been analyzed in [35, 36, 37, 38]. A classification of Android malware families based upon resource consumption over time can be found in [39]. Further, the battery drain caused by malicious (covert) software communication in mobile devices has been researched by Caviglione et al. [2]. Therein, the authors analyzed the energy consumption of android smartphones with and without malicious applications installed.

The mitigation of timing side channels has been investigated by Schinzel [40]. The author describes how timing-based side channels can be prevented due to the use of different methods, such as fixed runtimes or random delays. The optimization of ROC curves' AUC according to dynamic thresholds was analyzed in [41, 42].

3 Methodology

We investigate the time consumption of the computational intensive CC described in [1] in a controlled and encapsulated testing environment that is visualized in Figure 1. Therefore, we set up four *Raspberry Pi 4*, two with 2 GB RAM and 2 with 4 GB RAM. We decided not to use virtual machines because we wanted to perform our experiments within a physical network. The Raspberries with larger RAM were used for the roles consuming the most resources, i.e., OS (observation-point for packet runtime) and CR (multiple hash operations to reconstruct the original message). We connected all four systems via network cables to a *Fritzbox 7340* that served as a switch. For minimizing side effects, we disabled the wireless interface of the Fritzbox. To maintain the testing environment, the Wi-Fi interfaces of the Raspberries were connected via a wireless access point to a second network to minimize potential performance issues of the cable-bound interfaces, splitting the control traffic (Wi-Fi) and the produced data (cable). We utilized *Wireshark* and observed the runtime of each hash-containing packet leaving the `eth0` interface of the OS until the return of the associated ACK. All of our scripts were implemented with Python version 3. For sending and modifying packets we used the *Scapy* python library; for catching network traffic we used the python package *netfilterqueue* and the Linux built-in packet filter *iptables*.

3.1 Covert Channel Implementation

Our experiment depicts the man-in-the-middle (MitM) scenario, in which the CS and CR are passed through by network packets as described in [1]. First, we emphasize that our scripts were written with the focus on confining the time distinction to the utilized hash algorithm itself. Further, our code prevented disk writing and reading operations as they are time intensive and might falsify the results, so all necessary information were written into the RAM at the program startup. The sequence of the experiment is described in Figure 1.

In step **1 a)**, the OS sends a SYN packet to the OR. It contains the next OTP hash x_{n-1} that is

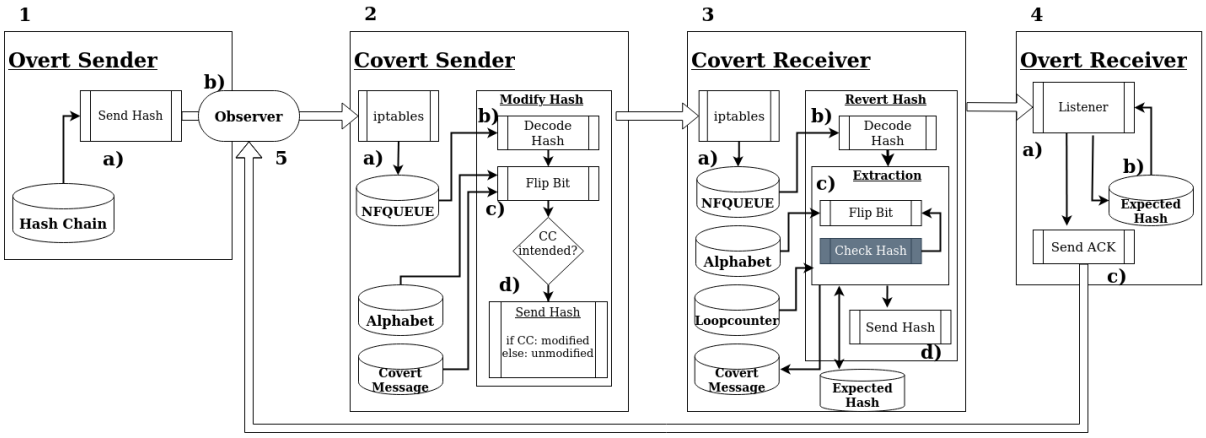


Figure 1: Structure of the experimental setup

necessary for successful authentication. We precalculated four OTP hash chains as described in section 2.1.1, one each based upon MD5, SHA2-384, SHA3-256 and SHA3-512. The final hash x_n was initially stored on the OR as described in section 2.1.1 and is represented in Figure 1 by the container *Expected Hash* (**4**). This hash is also known by the CR as it passed the CR heading to the OR. At the moment the packet leaves the OS, the observer is passed and the measurement for the current hash starts **(1 b)**. The CS catches the packet using an *iptables* rule and pushes the packet to the NFQUEUE table, represented by step **2 a)**. The python script we wrote fetches present packets from this table and in the first step decodes the contained hash into bits (**2 b)**). Now, the covert information is encoded into the bit-sequence. Therefore, first both the CS and the CR had to agree on an alphabet representing encoded information as described in section 2.1.2. We assumed that the quality of this alphabet would have influence on the runtime performance, so we defined two alphabets. The first one was the best-case alphabet, in which the most frequently used letter in the covert message is represented by the first bit flipped, followed by the second most frequently used letter and so on. This limits the subsequent hash operations that have to be performed by the CR to the necessary minimum. Second, we defined the worst-case alphabet in which the information were encoded the other way around, forcing the CR to consume the maximum time by computing hashes. The covert message used for our experiments was the short story *The Haunter Of The Dark*, written by H. P. Lovecraft. The information is encoded in step **2 c)**, followed by the forwarding of the hash to the CR (**2 d)**). Within this step, the CS decides whether it sends the modified hash ($x_{modified}$) or forwards the original unmodified hash (x_{n-1}). This decision depends on a parameter given at program startup, defining the CC-share in percent. The parameter range reaches from 0% (no hashes are modified) to 100% (all hashes are modified). As previously said, we focused on limiting the time-distinction to the hash operations performed on the CR. All operations on the CS were performed the same way for both, with and without CC.

The CR catches the hash-containing packet using an *iptables* rule and pushes it to the NFQUEUE table **(3 a)**. Identically to the CS, our script fetches the packet, extracts the hash $x_{modified}$ and decodes it to a sequence of bits **(3 b)**. After this process is completed, the CR successively flips each bit singularly after another. For each bit flipped, CR tests if it created the original hash x_{n-1} from the modified hash $x_{modified}$ by checking if $f(x_{n-1}) \rightarrow x_n$. If the test is passed, the CR extracts the covert information according to the utilized alphabet and replaces the expected hash x_n with x_{n-1}, preparing for the next authentication hash x_{n-2}. The computational intensive operation is highlighted in Figure 1 and is filled with dark color. As already mentioned, we tried to confine the difference of CC and non-CC traffic on this operation. To

accomplish this, we had to ensure the same number of loops were performed for both, with active CC and without CC. Thus, we predefined a file containing the number of loops that have to be performed to revert the covert message. The CR initially tests if the submitted hash has been modified and then decides whether to perform the hash operations, or replaces it with a `NOP` operation. This method was described for instance in [43] and we applied it with a simple `pass` in our python script, replacing the hash operation within each loop. After the process is finished, CR forwards the packet containing the original hash x_{n-1} to the OR. The OR accepts the packet with a service listening on port 42424 (**4 a)**. The service checks if the received hash is correct by performing $f(x_{n-1})$, comparing the result with the previous stored hash x_n. If the hash is correct, the OR replaces x_n with x_{n-1} to prepare for the next authentication request containing the hash X_{n-2}. This process is visualized in step **4 b)**. Finally, if the test is passed, the OR sends an acknowledgment back to the OS (**4 c)**. The Observer catches this acknowledgment and finishes the measurement for each packet (**5**).

3.2 Data Generation

We decided to use the hash-functions SHA3 with the bit lengths of 512 and 256, as well as SHA2-384 and MD5 for our hashing operations, as their consumption of computation time and their hash value length differs. We assume SHA3-512 will lead to larger packet runtimes and thus better detectability compared to the other utilized hash-algorithms. In sum, we investigated 8 scenarios in this paper:

1. SHA3-512 with the best alphabet
2. SHA3-512 with the worst alphabet
3. SHA3-256 with the best alphabet
4. SHA3-256 with the worst alphabet
5. SHA2-384 with the best alphabet
6. SHA2-384 with the worst alphabet
7. MD5 with the best alphabet
8. MD5 with the worst alphabet

For each of these scenarios, we applied three experiments to visualize the time difference between an implemented CC and a CC-free communication. The first experiment creates the reference runtime without an encoding of covert information. This will further also be called *reference*. The second experiment is the implementation of a full CC, in which each hash was altered and carries covert information. This will further on also be called *full-CC* scenario. Finally, the third experiment alters every other hash to visualize the development of runtime with a shrinking share of covert information. This further on will also be called *half-CC* scenario. To produce enough data, we created a hash chain containing 5,000 hashes for each hash algorithm. These hashes were sent one after another from the OS via CS and CR to the OR and finally back to the OS. Between each hash sent, we applied a sleep-timer of one second to ensure the previous circle finishes before sending the next authentication request. Further, to deny random fluctuation of packet runtime, we repeated all experiments for each scenario ten times. In total, we performed 240 experiments, each lasting 1 hour and 30 minutes, i.e., 360 hours overall, resulting in 1,200,000 flows. For each flow carrying covert information, one symbol of the in advance agreed on alphabet has been transferred. I.e., after 5,000 flows performed for the full-CC and half-CC, 5,000 and 2,500 symbols of the alphabet have been transferred, respectively.

4 Evaluation

We observed the request-reply runtime of hash-based OTP authentication requests. The experiments were carried out as described in section 3.2 and the observation has been performed by the OS. In the following sections, we first determine the statistical significance of our data in Sect. 4.1. Afterwards, we present the results of our runtime evaluation in Sect. 4.2. Further, we investigate the detectability for each scenario after 5,000 packets in Sect. 4.3 as well as depending on the number of packets in Sect. 4.4. Finally, we discuss the limitations of our approach and countermeasures against it in Sect. 4.5.

Scenario	**SHA3-512**				**SHA3-256**			
	best alphabet		worst alphabet		best alphabet		worst alphabet	
	half-CC	full-CC	half-CC	full-CC	half-CC	full-CC	half-CC	full-CC
p-value	0.0415	$1.46 \cdot 10^{-14}$	$1.65 \cdot 10^{-31}$	$6.02 \cdot 10^{-94}$	0.3753	$0.74 \cdot 10^{-17}$	$3.65 \cdot 10^{-4}$	$1.81 \cdot 10^{-155}$

Table 1: Mann-Whitney-U test results for SHA3-512 and SHA3-256

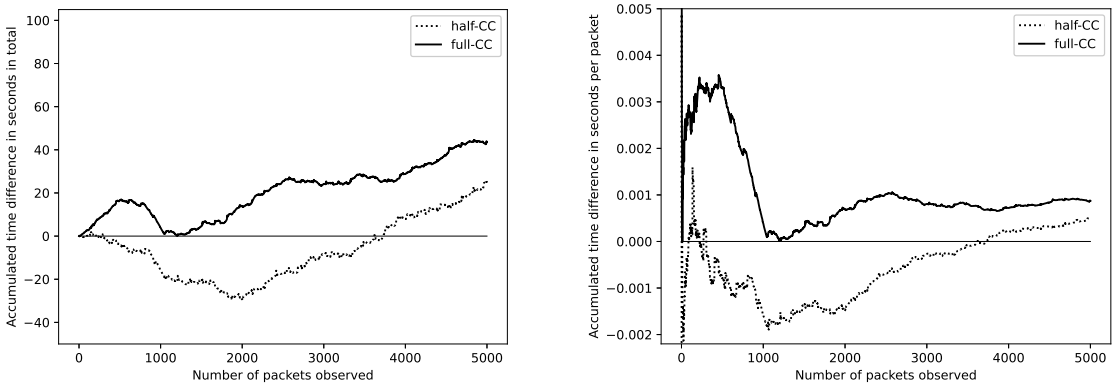
Scenario	**SHA2-384**				**MD5**			
	best alphabet		worst alphabet		best alphabet		worst alphabet	
	half-CC	full-CC	half-CC	full-CC	half-CC	full-CC	half-CC	full-CC
p-value	$2.34 \cdot 10^{-10}$	$1.94 \cdot 10^{-7}$	$2.38 \cdot 10^{-6}$	$3.87 \cdot 10^{-55}$	0.1637	$1.42 \cdot 10^{-6}$	$5.14 \cdot 10^{-7}$	$9.89 \cdot 10^{-73}$

Table 2: Mann-Whitney-U test results for SHA2-384 and MD5

4.1 Statistical Significance

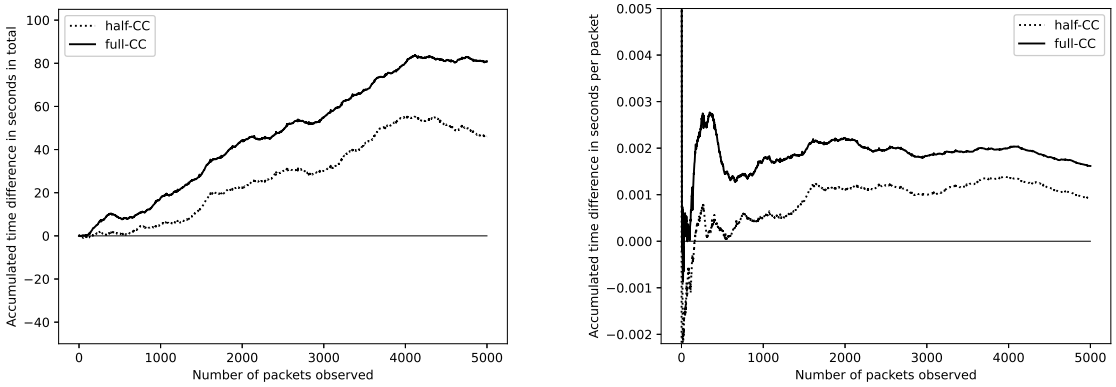
To ensure the statistical significance of our data, we performed Mann-Whitney-U tests. We decided to take advantage of this test because it compares a single ordinal variable without specific distribution of two data-sets to each other. The test results in the so-called p-value. The lower the value, the more statistically significant the difference. The data is commonly accepted to be statistically significant if $p<0.05$, further also called significance threshold. We compared the data of the covert information-transferring experiments to the corresponding none-CC experiments.

The results for both SHA3-256 and SHA3-512 are visualized in Table 1. The p-value for the half-CC implementation under the utilization of SHA3-512 with the best alphabet is 0.0415, which is slightly below 0.05. So, we assume the statistical significance is just acceptable. The results of all other SHA3-512 scenarios are far below the significance threshold. For SHA3-256 the significance value of the half-CC implementation with the best alphabet is at 0.3753. This may indicate that this scenario cannot be considered as statistical significant, i.e., this CC might remain undetectable. The SHA3-256 best-case alphabet full-CC scenario and both worst case alphabet scenarios remain below the value of 0.05 and can be considered to be statistical significant. The Mann-Whitney-U test results for SHA2-384 and MD5 are presented in Table 2. For SHA2-384 all values remain below the threshold and can be considered to be statistical significant. In the MD5 scenarios, the p-value is also clearly below the significance threshold, except for the half-CC best alphabet implementation. This value at 0.1637 indicates that the results for the MD5 half-CC scenario utilizing the best alphabet, cannot be considered statistically significant. Thus, we can expect that one of our experiments might remain undetectable. We assume the loss of statistical significance for both the SHA3-256 and MD5 best alphabet with half-CC scenario is caused by the combination of the less time-consuming hash operations in comparison to SHA3-512 and the fewer hash operations performed due to the best-case alphabet. We expect the significance to increases with the number of performed hash operations.



(a) Accumulated time difference of all 10 measurements (b) Accumulated time difference per packet

Figure 2: SHA3-512 implementation with the best alphabet



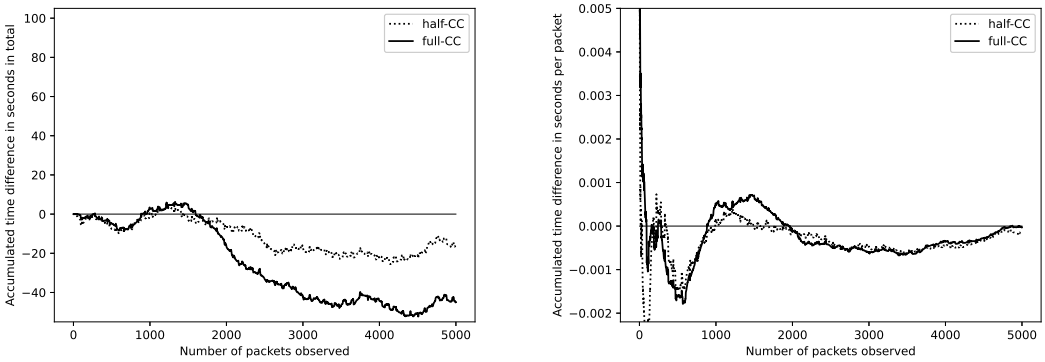
(a) Accumulated time difference of all 10 measurements (b) Accumulated time difference per packet

Figure 3: SHA3-512 implementation with the worst alphabet

4.2 Runtime Evaluation

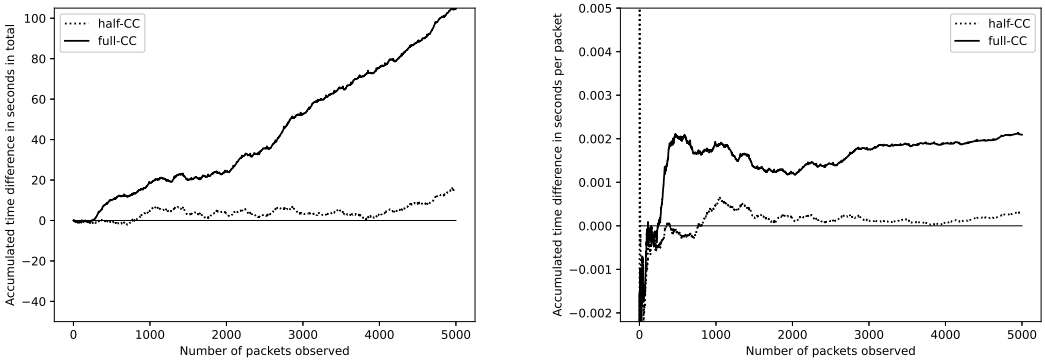
Although we have focused on eliminating external influences, the packet runtime is still influenced by some random events, for example by the scheduler of the operating system. To minimize such effects, we accumulated the request-reply timing of all experiments performed for one specific scenario. We further compared the accumulated time of the experiments without CC to the accumulated time of the experiments carrying covert information to investigate the difference. For each scenario, we created two plots. Plot *a* shows the accumulated difference of runtime in total seconds for 10 accumulated experiments, compared to the accumulated 10 reference measurements without CC. Plot *b* shows the time difference per packet averaged over all experiments of a CC scenario compared to value averaged over all reference experiments.

SHA3-512 Figure 2(a) shows the accumulated results of the SHA3-512 best alphabet experiments. The full-CC implementation, represented by the solid line, is permanently above the reference, i.e., the accumulated request-reply cycle has consumed more time than the accumulated runtime of the non-



(a) Accumulated time difference of all 10 measurements (b) Accumulated time difference per packet

Figure 4: SHA3-256 implementation with the best alphabet

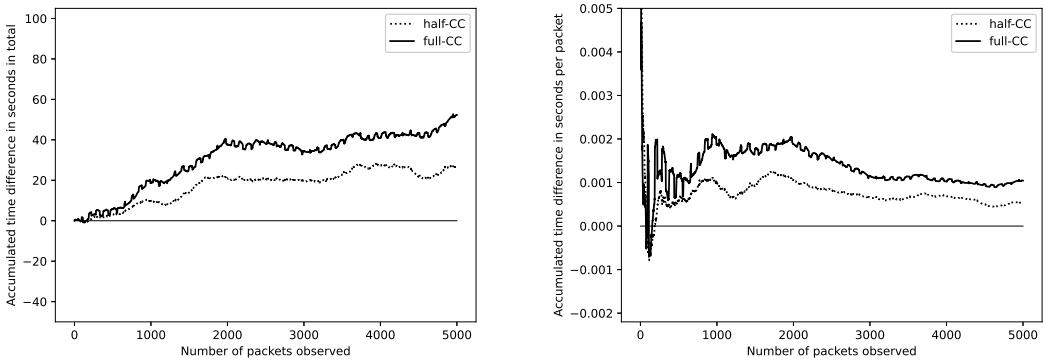


(a) Accumulated time difference of all 10 measurements (b) Accumulated time difference per packet

Figure 5: SHA3-256 implementation with the worst alphabet

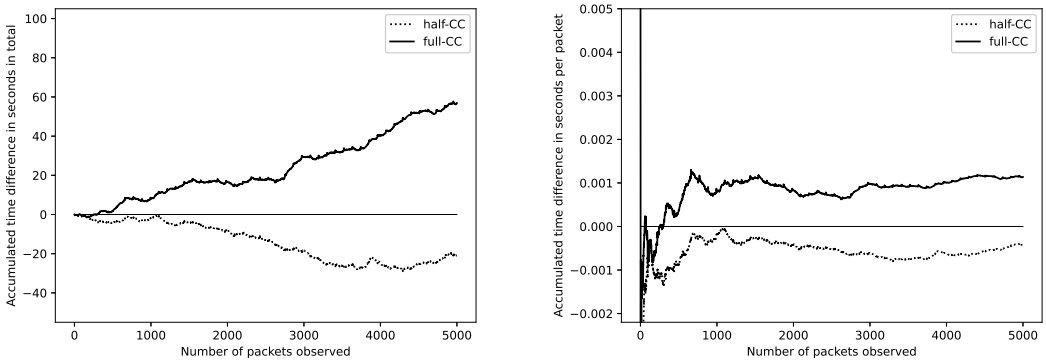
CC experiments. The half-CC scenario remains below the reference until approximately 3,500 flows, consuming less time in total until this point. Finally, the accumulated request-replies consume more time than the reference implementation. Figure 2(b) shows the same results specified for the mean request-reply runtime of one single flow. Again, the solid line represents the full-CC implementation. The time difference per packet, in the beginning, is clearly above the reference implementation and stabilizes at about 0.001 seconds more time consumed. The half-CC implementation consumes less time in the beginning compared to the reference implementation but finally turns into the positive area and finishes at 0.0005 seconds consumed additionally per packet than the reference.

The results for the scenario SHA3-512 with the worst alphabet are presented in Figure 3. Both CC scenarios permanently consume more accumulated time than the reference (Figure 3(a)). The time-consumption difference per packet is visualized in Figure 3(b). The full-CC first resides around 0.0002 seconds and finally stabilizes slightly below this value. The half-CC first resides like the full-CC implementation and stays constantly between the reference and the full-CC. All figures created for SHA3-512 show the important influence of the utilized alphabet. Further, the figures show that the lower the share of CC packets within an implementation, the harder it is to detect. This is especially an issue if the CC is realized with a perfect alphabet.



(a) Accumulated time difference of all 10 measurements (b) Accumulated time difference per packet

Figure 6: SHA2-384 implementation with the best alphabet

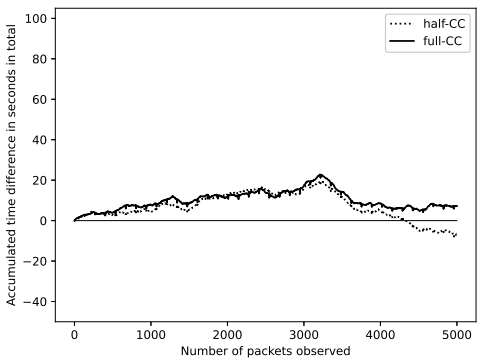


(a) Accumulated time difference of all 10 measurements (b) Accumulated time difference per packet

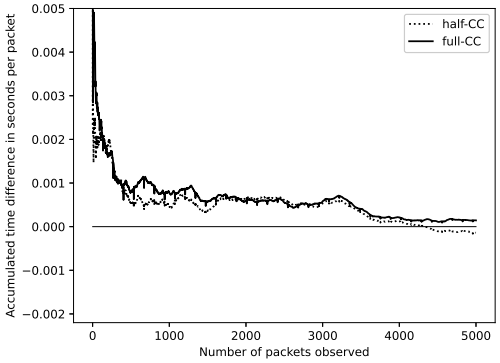
Figure 7: SHA2-384 implementation with the worst alphabet

SHA3-256 The results for the scenario SHA3-256 with the best possible alphabet are presented in Figure 4. The accumulated total runtime shown in Figure 4(a) indicates that both full- and half-CC consume less time in total than the reference. The runtime is equal to the reference until 2,000 packets and then differs clearly for both full- and half-CC. Although, the total results seem to be clear, the result differs only slightly if set relating to the number of packets, leading to almost the same runtime per packet. The results are influenced by the few hash operations performed due to the optimized alphabet. Further, please note the Mann-Whitney-U test performed for the SHA3-256 half-CC scenario with the best alphabet was not considered to be statistical significant. The result might be falsified by this fact. The results for the implementation of SHA3-256 with the worst possible alphabet are clearer, and the total runtime differs noticeably for the full-CC (Figure 5(a). The total runtime increases constantly and reaches a maximum of 100 seconds after 5,000 packets. The half-CC consumes slightly more time and stays near the reference over the entire duration of the measurements. This is underlined by the runtimes set relating to the number of packets (Figure 5(b)).

SHA2-384 The experiments performed with SHA2-384 and the best alphabet are presented in Figure 6. Both the total runtime and the runtime relating to a single packet rely on a constant value. It is also visible

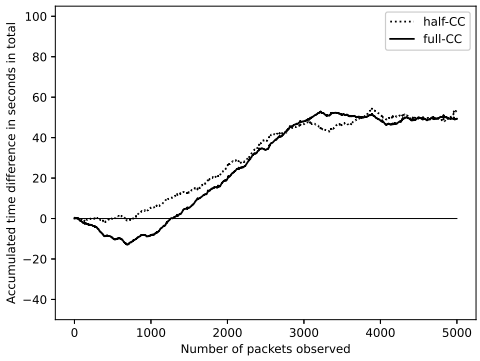


(a) Accumulated time difference of all 10 measurements

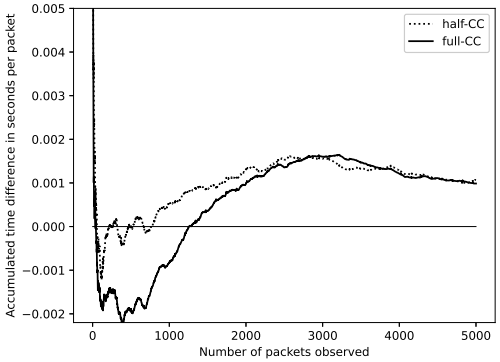


(b) Accumulated time difference per packet

Figure 8: MD5 implementation with the best alphabet



(a) Accumulated time difference of all 10 measurements



(b) Accumulated time difference per packet

Figure 9: MD5 implementation with the worst alphabet

that the full-CC consumes more time than the half-CC implementation. The runtime per packet differs by 0.001 seconds for the full-CC and 0.0005 seconds for the half-CC compared to the reference experiments. For the worst possible alphabet, the total runtime of the full-CC experiments clearly consume more time than the reference experiments and increases continuously. The half-CC experiments remain around the reference until 2,500 packets and then seem to consume less time, but the total runtime remains only slightly below the reference (Figure 7(a)). In relation to the runtime of a single flow, the difference of the half-CC experiments remains nearly constantly at 0.0005 seconds below the reference experiments, while a single packet of the full-CC consumes constantly 0.001 seconds more time compared to a single packet of the reference. This indicates lower potential detection rates for the half-CC scenario, which might remain undetectable.

MD5 The accumulated time consumption of the MD5 scenario implementing the best possible alphabet is visualized in Figure 8(a). Again, we compared the accumulated time consumption of the CC experiments with the accumulated results of the reference experiments. The figure shows that the MD5 hash algorithm consumes less time than all CCs with SHA3-512 and SHA2-384. Merely, the SHA3-256 algorithm with full-CC and half-CC consumes less time. In the beginning, both MD5 best alphabet CC-

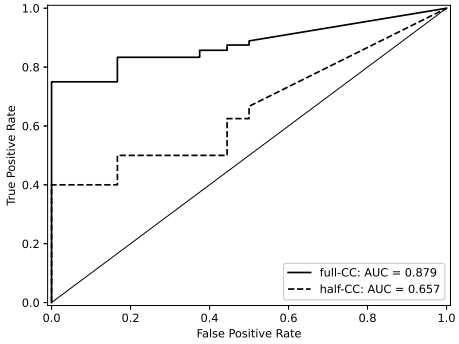
implementations constantly consume more time than the reference, but at some point the time difference for the full-CC shrinks to a barely observable value.

Similarly for the half-CC experiments, which

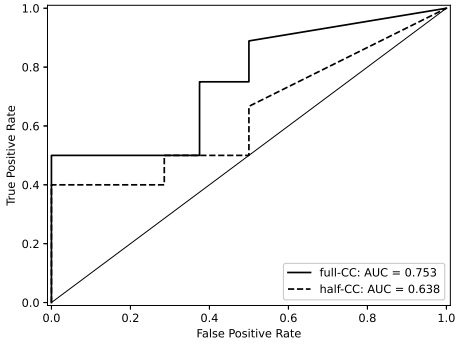
even have a better performance than the accumulated reference experiments in the end. The average request-reply difference per packet underlines these results and is presented in Figure 8(b). In the beginning, the difference of both CC scenarios remains constant at about 0.0005 seconds. Finally, the time difference shrinks and resides around the average reference packet runtime. Since the runtime of the packets slightly depends on some random factors like the operating system schedulers of CS, CR and OR, we assume the time-consumption of MD5 under the utilization of the best alphabet is too similar to the reference for a clear distinction. This is compliant with our previous significance test.

The ratio of the accumulated MD5 experiments under the use of the worst possible alphabet is visualized in Figure 9(a). In the beginning, the full-CC performs superior to the reference consuming less time. In the further progression, the accumulated packet request-reply cycles of the full-CC experiments consume more time than the accumulated reference cycles. Finally, the accumulated time difference reaches a plateau and stays constant. The accumulated time of the half-CC experiments behaves just like the accumulated time of the full-CC experiments, except it is pending around the x-axis at the beginning instead of peaking into the negative area. Setting the consumed time in relation to a single packet underlines this observation and is presented in Figure 9(b). An average single packet of the full-CC experiments consumes less time in the beginning and resides around 0.002 seconds. In the further progression, each packet consumes successively more time and the average runtime of the full-CC per packet increases to 0.001 seconds more consumed than the reference. The same observation can be made for the half-CC experiments, except that the time pends around the x-axis at the beginning instead of consuming less time than the reference.

Discussion The interpretation of the figures leads to the conclusion that it is more challenging to detect a computational intensive reversible CC if a resource-efficient hash function is utilized. This especially can be observed for MD5 and SHA3-265 with the best possible alphabet. Also, the worst-case alphabet packet runtimes of the MD5 and SHA2-384 CC experiments suggest this conclusion. The SHA3 CC implementations are more time-consuming, especially for non-optimal alphabets. The utilized alphabet is even more crucial to our detection approach than the resource efficiency of the hash algorithms. The runtimes of both full-CC and half-CC experiments with the worst alphabet are more distinct to the reference than all best alphabet experiments for all algorithms and hash lengths. Especially the MD5 worst alphabet runtimes for both full-CC and half-CC as well as the SHA2-384 full-CC runtime support this observation. All of our experiments were performed with 5,000 flows. The per-packet evaluation of all scenarios shows that differing runtimes for CC implementations can be observed after a different number of flows. In Fig. 2(b), for the full-CC experiments, the per-packet runtime differs after approximately 2,500 flows and stays constant until 5,000 flows. For the worst alphabet this can be observed after less than 1,000 flows (Fig. 3(b)). For the worst alphabet scenario of MD5 (Fig. 9(b)) and the best alphabet scenario of SHA2-384 (Fig. 6(b)), it can be observed that the runtime of the CC implementations approaches to the reference again at the end of our experiments. We selected the value of 5,000 flows for our paper to evaluate if a packet-runtime based approach is conceivable. Our results indicate, this approach is feasible, though further research has to be performed. Long-lasting experiments with more than 10,000 flows would potentially further clarify the impact of the recording duration (and if more than 5,000 flows have to be recorded for improved results).

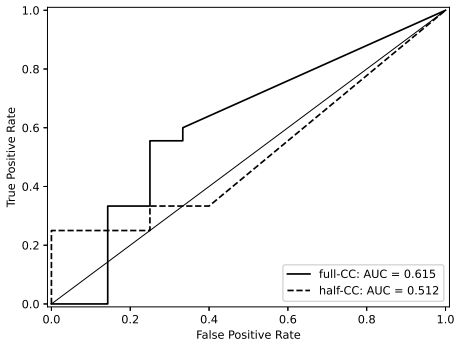


(a) ROC chart of SHA3-512 with the best alphabet

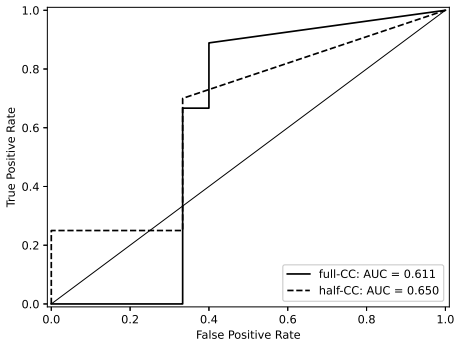


(b) ROC chart of SHA3-512 with the worst alphabet

Figure 10: Threshold-based detection-rate of the SHA3-512 experiments



(a) ROC chart of SHA3-256 with the best alphabet



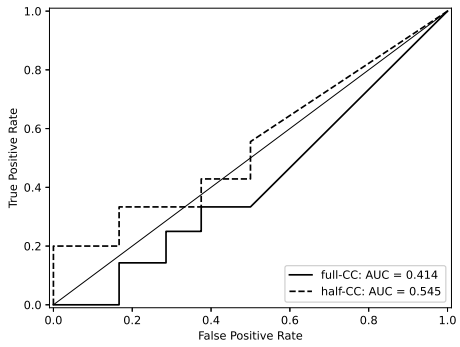
(b) ROC chart of SHA3-256 with the worst alphabet

Figure 11: Threshold-based detection-rate of the SHA3-256 experiments

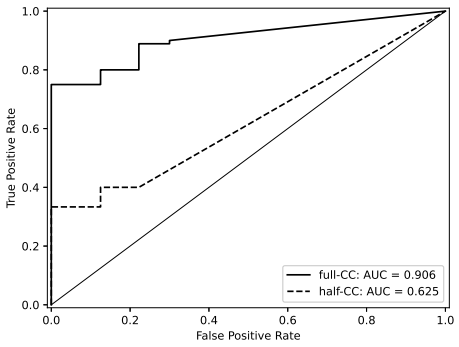
4.3 Threshold-based Detection

Our proposed detection approach utilizes a threshold, defined by the averanged packet-runtime of reference traffic and is applied in this section for the entire observation period of 5,000 packets. For each scenario, we applied the averaged runtime per packet of all reference experiments after 5,000 flows as the metric. We compared the average runtime per packet of every single CC experiment after 5,000 flows to this value. Afterwards, we performed a threshold-based test for separate groups (also for now on called detection setups), splitting the full-CC and half-CC experiments. This was decided to determine if the detection rate differs if the flows contain a lower share of CC traffic. A detection setup contained all 10 reference measurements as well as all 10 dedicated experiments performed with the full-CC or the half-CC implementation. For each experiment in a detection scenario, we compared the packet runtime to the threshold. If the average packet runtime was above the threshold, an active CC was assumed; if the value was below the threshold, legitimate traffic was assumed.

For each scenario, we created a ROC curve to visualize the detection rate. To create the curves, so called partitions with are created, that compare the true positive rate (**TPR**) and false positive rate (**FPR**)each partition to the following partitions. The rates are calculated by the formulas

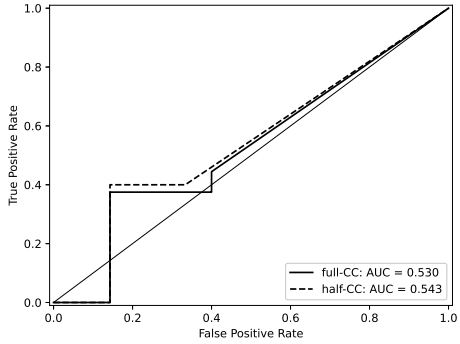


(a) ROC chart of SHA2-384 with the best alphabet

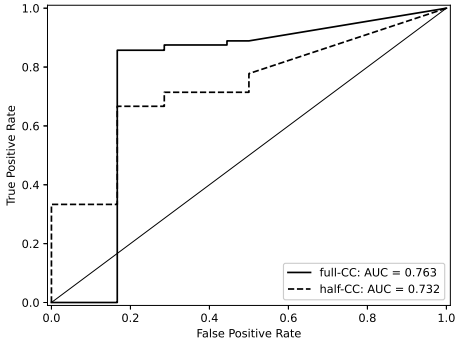


(b) ROC chart of SHA2-384 with the worst alphabet

Figure 12: Threshold-based detection-rate of the SHA2-384 experiments



(a) ROC chart of MD5 with the best alphabet



(b) ROC chart of MD5 with the worst alphabet

Figure 13: Threshold-based detection-rate of the MD5 experiments

$$\text{TPR} = \frac{\mathit{True\ Positive}}{\mathit{True\ Positive} + \mathit{False\ Negative}}$$

$$\text{FPR} = \frac{\mathit{False\ Positive}}{\mathit{False\ Positive} + \mathit{True\ Negative}}$$

Figure 10(a) shows the results for the **SHA3-512** scenario with the best possible alphabet. The full-CC detection scenario's *Area Under Curve* (AUC) is 0.879, which can be considered as good. For the half-CC experiments, the AUC is 0.657. This value indicates that this scenario is only marginally detectable as the value is near above tossing a coin. The threshold-based detection result of the scenario SHA3-512 with the worst possible alphabet is presented in Figure 10(b). The AUC value for the full-CC detection scenario is at 0.753 and indicates a moderate detectability. The AUC value of the half-CC experiments with the worst possible alphabet is almost the same as the AUC value of the half-CC experiments with the best possible alphabet. The AUC would indicate a better detectability for the best alphabet, which opposes the results presented in the previous section. We investigated this observation further and can reduce it to the appearance of a false negative in the first partition of the SHA3-512 worst alphabet detection scenario. This is misleading due to the decreased FPR for the worst alphabet, leading

to a lower AUC than for the best alphabet. We assume the AUC will clarify if more experiments are performed. According to our extended investigation described above, we assume both SHA3-512 full-CCs of each alphabet are nearly as detectable as each other, however the AUC differs clearly. Because of the increased time difference between CC-implementations and the reference, we expected to have higher detection rates due to the increased runtime with the implemented worst-case alphabet as visualized in Figure 3(b) compared to Figure 2(b).

The results for **SHA3-256** with the best possible alphabet are presented in Figure 11(a). Please note that the results for the half-CC implementation cannot be considered statistical significant. Matching the insignificant difference, the AUC is at 0.512, which equates to coin flipping and can be considered as undetectable. The detection rate for the full-CC is nearly the same as for the half-CC implementations with SHA3-512. The AUC is at 0.615, which indicates a marginal detectability. The results for the worst-case alphabet are presented in Figure 11(b). The AUC for both full-CC and half-CC detection setup equate nearly to the results for the best alphabet. Though, the half-CC AUC is higher and the results are statistically significant. We explain the decreased rates to the fewer time-consuming hash-operations that are performed with SHA3-256 in comparison to SHA3-512.

How crucial the alphabet is, can also be observed in Figure 12(a) that presents the results for the detection scenario **SHA2-384** with the best alphabet. Both full-CC and half-CC detection rates equate to guessing. This result is also interesting since the time consumption of the CC experiments, as presented in Figure 6, clearly differs from the reference experiments. Single CC experiments seem to have an influence on the accumulated runtime, but are only detected once. The detection rate of the full-CC with the worst alphabet SHA2-384 detection scenario is the best as the AUC is at 0.906 (Figure 12(b)). The AUC for the half-CC experiments is slightly above 0.6 and indicates a low detectability, like the runtime comparison in the previous section already suggested.

The ROC curves for the **MD5** detection scenario can be found in Figure 13. The detection rate of the best alphabet experiments nears to tossing coins and can be assumed to be undetectable with our approach (Figure 13(a)). This expected result can be explained by the lack of statistical significance, the utilized best-case alphabet, and the utilization of the fewer resource-consuming MD5 hash algorithm. The worst-case implementation in contrast can be assumed as moderately detectable with an AUC value of 0.763 for the full-CC scenario and 0.732 for the half-CC scenario.

For all detection scenarios, we observed mostly good detection rates with the worst possible alphabet implemented combined with a full-CC altering all hashes. One exception is the SHA3-256 worst alphabet detection scenario that resulted in moderate detection rates. The half-CCs with the worst alphabet are also detectable, but the rate is clearly lower than only 50 percent of the hashes were altered and consume more time than the reference. For the best possible alphabet, only SHA3-512 was reliably detectable. The other hash-algorithms seem to consume too little time for good detection rates. We can conclude that both the utilized algorithm and the utilized alphabet are crucial for a threshold detection based on packet runtime.

4.4 Detection Rate Depending On The Number Of Packets

In the previous subsection, the proposed threshold-based runtime detection approach has been applied to identify CC activity after 5,000 flows by evaluating our previously described experiments. However, the detector can also be applied after fewer than 5,000 observed packets. Therefore, we created figures for each detection scenario, displaying the AUC value after each 50 packets from 50 to 5,000 packets. The respective threshold is equivalent to the number of the average packet runtime of all according reference experiments, i,e., after 50 flows, 100 flows and so forth. Besides the AUC Value of ROC Curves, there exist further statistical metrics to determine, whether a detector is performing well. In addition to the AUC, we will further present the detectors precision, recall and accuracy for each detection scenario.

The metrics evaluate the deviation of True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN) by calculating the distribution in different means and can visualize the quality of a detector. The **precision** metric describes the number of TP compared to all positives and describes the detection rate of actually CC containing experiments and is calculated as follows:

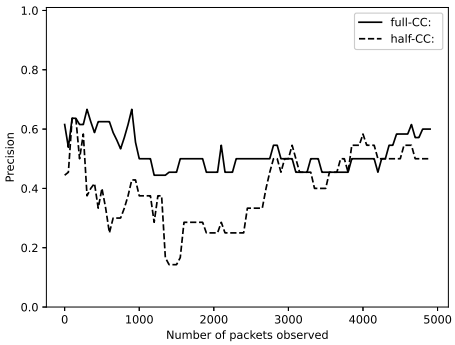
$$\text{precision} = \frac{TP}{TP+FP}$$

The **recall** metric describes the TP rate in relation to all values that are considered to be positive, i.e., TP and FN, and indicates how well a detector can determine true positives correctly. The recall is calculated by the formula

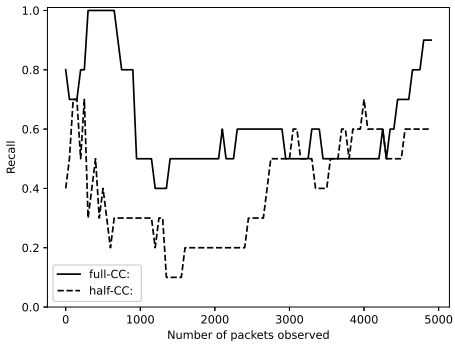
$$\text{recall} = \frac{TP}{TP+FN}$$

The **accuracy** metric calculates the share of correct classified experiments and compares the TP and TN to the total number of classified experiments, i.e., TP, TN, FP and FN. The accuracy can be applied to our data, because the number of positive and negative samples is balanced for each detection scenario.

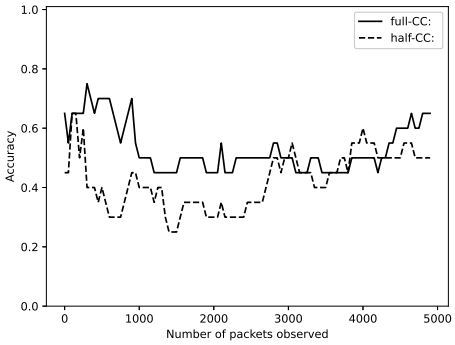
$$\text{accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$



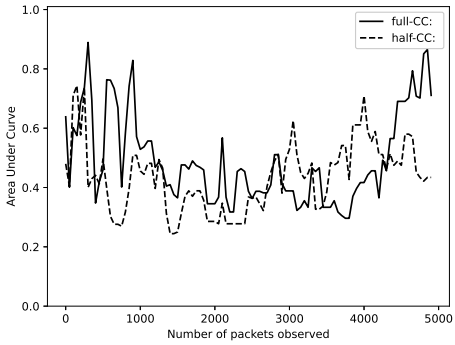
(a) Precision



(b) Recall

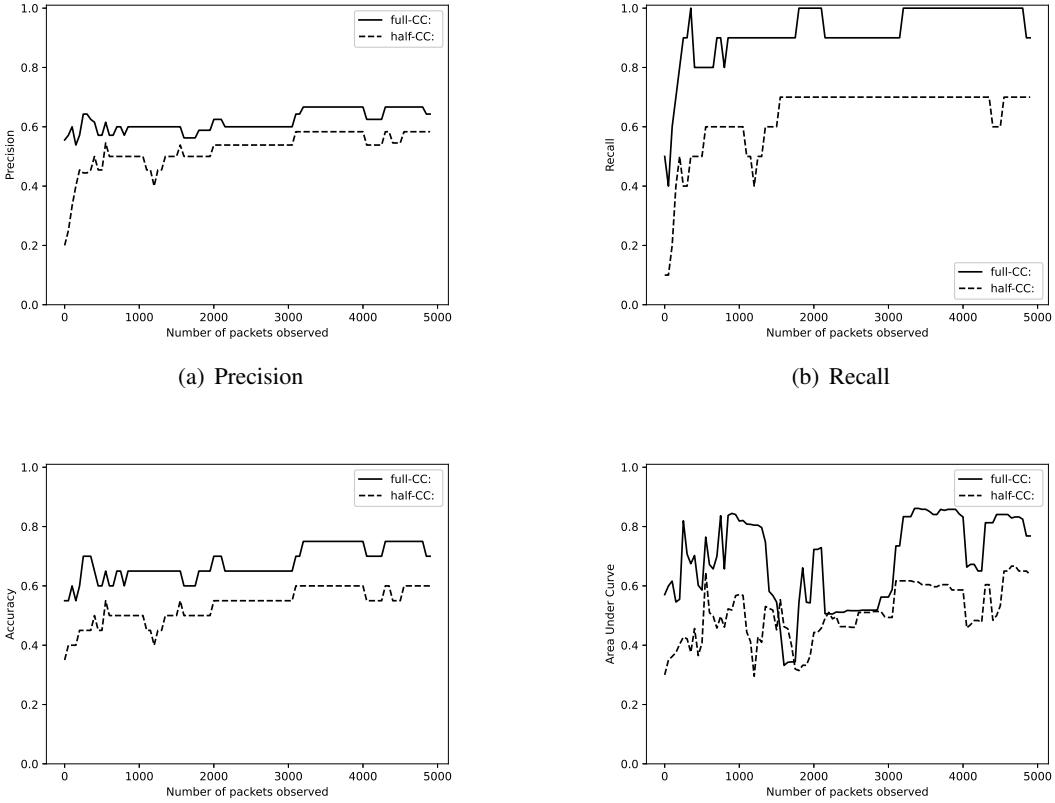


(c) Accuracy



(d) Area Under Curve

Figure 14: SHA3-512 best alphabet: Statistical metrics according to the number of packets



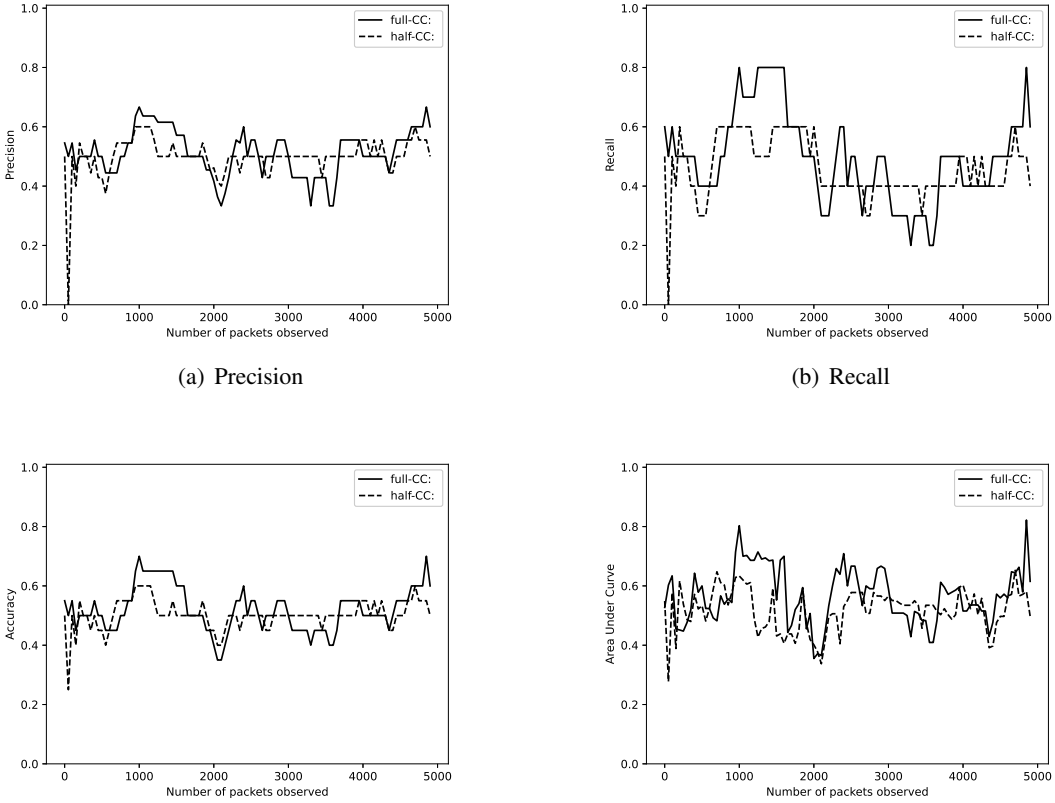
(a) Precision (b) Recall

(c) Accuracy (d) Area Under Curve

Figure 15: SHA3-512 worst alphabet: Statistical metrics according to the number of packets

SHA3-512 The results of the SHA3-512 best alphabets are presented in Fig. 14. The precision remains below or slightly above 0.5 for the entire observation period. This indicates moderate false positive rates that will raise false alerts. The recall metric of this detection setup illustrates, that until 4,000 packets that half and more than half of the CC experiments were not correctly detected for the full-CC and half-CC, respectively. Though, the value before 1,000 flows indicates a well-working detector for full-CCs, the detector cannot be considered to be reliable. After 4,000 flows the threshold seems to increase for the full-CC scenario and seems to achieve good correct detection rates. Overall, the accuracy indicates, that only half of the experiments were classified correctly for both, full-CC and half-CC, however the value increases for the full-CC scenario after 4,000 flows indicating better results. Finally, the AUC underlines these observations, though the performance of the detection increases after 4,000 packets and might generate good results. A share of 50 percent CC containing flows might remain undetectable for the combination of SAH3-512 and best alphabet.

The results for SHA3-512 and the worst alphabet are presented in Fig. 15. Compared to the previously in Fig.14 presented results, the values indicate better detection rates. The precison is clearly improved for the half-CC experiments and slightly improved for the full-CC experiments, i.e., fewer false positives are detected. Further, the recall illustrates good detection rates for both detection scenarios. Full-CCs are constantly reliable detected after 400 flows, half-CCs after 1,750 flows. This is underlined by the accuracy, that indicates that constantly more than 60 percent and 50 percent of all full-CC and half-CC experiments were categorized correct, respectively. The AUC value stabilizes after



(a) Precision

(b) Recall

(c) Accuracy

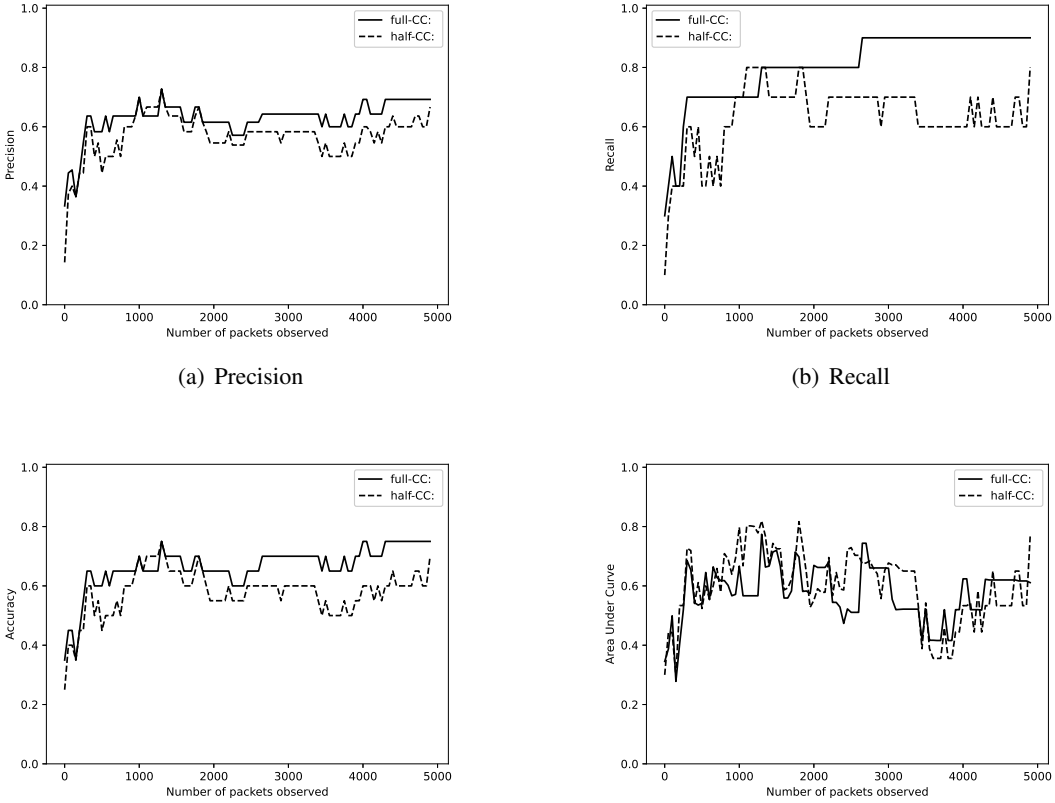
(d) Area Under Curve

Figure 16: SHA3-256 best alphabet: Statistical metrics according to the number of packets

3,000 flows for the full-CC scenario and relies at around 0.5 for the half-CC scenario.

SHA3-256 The detection metrics depending on the number of packets for the SHA3-256 experiments with the best alphabet are visualized in Fig. 16. All presented values let us conclude both, the full-CC scenario and half-CC scenario, remain undetectable, if performing with a perfect alphabet. The precision metric for full-CC and half-CC remain constantly around 0.5 for all packets observed. The recall value is pending around 0.5, which also indicates undetectability, like the precision metric. The accuracy of both scenarios remains at around 0.5 for nearly the complete runtime and also indicates detection results that are comparable to coin-flipping.

In contrast to the results of the best alphabet, the metrics of SHA3-256 with the worst alphabet show that our detection approach produces acceptable results (Fig. 17). The precision stays constantly above 50 percent after 500 packets and indicate the number of false positives is utilizable, but not good. The recall indicates low false negative rates. For the full-CC detection scenario, good results are achieved after 500 packets and for the half-CC the number of FN is acceptable after 1,000 packets. For both scenarios, the accuracy, i.e., the number of correctly categorized experiments is constantly around 0.75 and around 0.5 for full-CC and half-CC, respectively. The AUC value is mostly above 0.6 for both scenarios, however the value decreases after around 3,000 packets and stabilizes again after 4,000 packets. As the AUC relies on recall and false positive rate and the recall metric presented in Fig. 17(b) indicates good results, it can be concluded that at this point the detector produces high false positive rates.



(a) Precision

(b) Recall

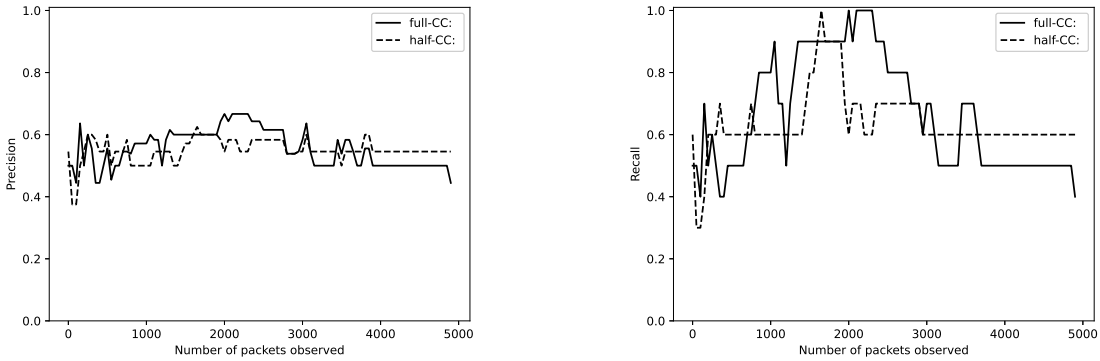
(c) Accuracy

(d) Area Under Curve

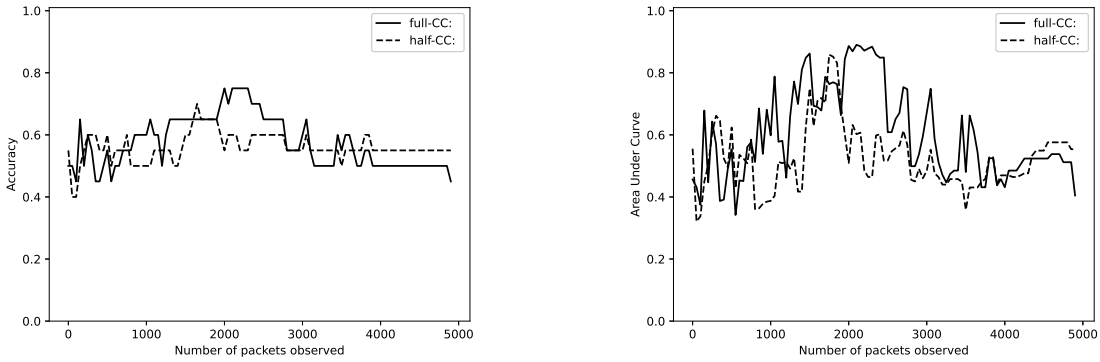
Figure 17: SHA3-256 worst alphabet: Statistical metrics according to the number of packets

SHA2-384 The results for the best alphabet detection scenarios are presented in Fig. 18. For both, precision and accuracy, the value is constantly slightly above 0.5 for full-CC and half-CC experiments. Both charts indicate, that half of the true positives and true negatives has been correctly categorized compared to the number of FP in case of precision and total experiments in case of accuracy. For both, the recall and AUC metric, the results between 1,000 and 3,000 packets seem to be fine, however, the values deteriorate after 3,000 packets and remain at 0.5. Like for other best alphabet detection scenarios, the CC traffic cannot be considered to be correctly detected but it seems like the number of true negatives might be low in between 1,000 and 3,000 flows.

Like for SHA3-512 and SHA3-256, the results of the worst possible alphabet under utilization of SHA2-384 (Fig. 19) indicate higher detection rates compared to the best alphabet if every hash has been altered. The precision of our detection approach achieves good results for the full-CC experiments after 500 observed packets while the results for the half-CC experiments remain low and are pending around 0.5, indicating a high number of FPs. The recall metric of the full-CC also indicates a low number of FNs. However, the half-CC detection setup remains below 0.4 and indicates a high number of FNs. The same can be observed for the accuracy, which is constantly performing well for the full-CC detection, but only acceptable for the half-CC detection. The AUC indicates clearly, that the full-CC will be detectable by our approach after 500 flows. However, the half-CC with the worst alphabet will potentially remain undetectable within the first 5,000 packets.



(a) Precision (b) Recall



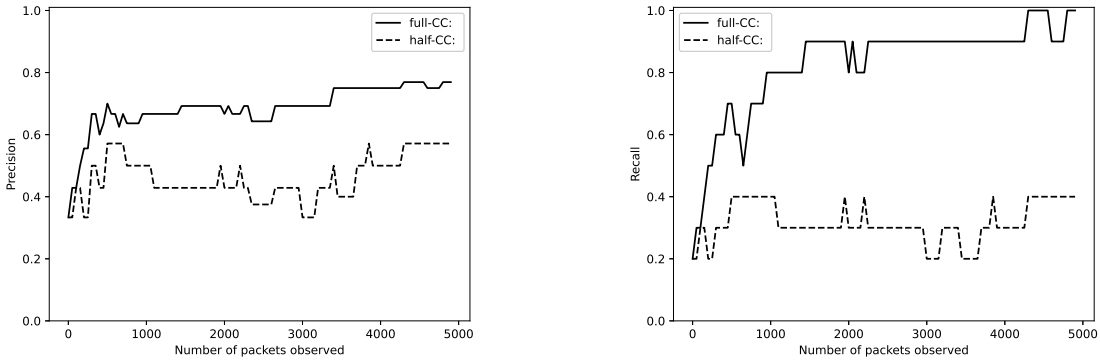
(c) Accuracy (d) Area Under Curve

Figure 18: SHA2-384 best alphabet: Statistical metrics according to the number of packets

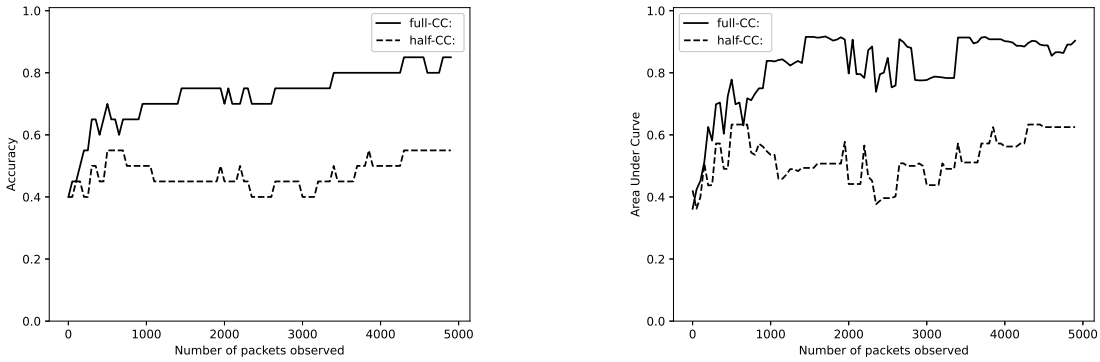
MD5 The statistical metrics for the MD5 best alphabet detection scenario are presented in Fig. 20 and indicate, that the utilization of the best alphabet will impede a detection with our proposed approach. The values of precision, recall, accuracy and the AUC are constantly remaining or pending slightly above 0.5, for both full-CC and half-CC. According to these results, a detection will not be possible.

While the best alphabet prevents the detection, the metrics of the MD5 detection setups under the utilization of the worst alphabet indicate good detection results after 1,000 packets for both full-CC and half-CC (Fig. 21). The precision stays constantly slightly below 0.7 and implies acceptable FP rates. Good results are also pointed out by the recall metric after 1,000 and 2,500 packets for the full-CC and half-CC, respectively. This states out the TP rate compared to the sum of detected CCs is over 0.75 for the full-CC after 1,000 packets and increases to 0.9 after 3,000 flows. The half-CC TP rate is acceptable after 2,000 packets and can be considered as 'good' after 2,700 packets. Further, the accuracy expresses a correct classification of more than 75 percent of all performed experiments for both detection scenarios. Finally, the AUC indicates good detection rates after 1,000 packets for the full-CC and for the half-CC after 2,700 flows.

Discussion The detection-rate evaluation underlines once more how crucial the chosen alphabet is for our detection approach. For all researched detection scenarios implementing the *best* possible alphabet, our proposed detection approach has only resulted in good detection rates for the SHA3-512 full-CC



(a) Precision (b) Recall



(c) Accuracy (d) Area Under Curve

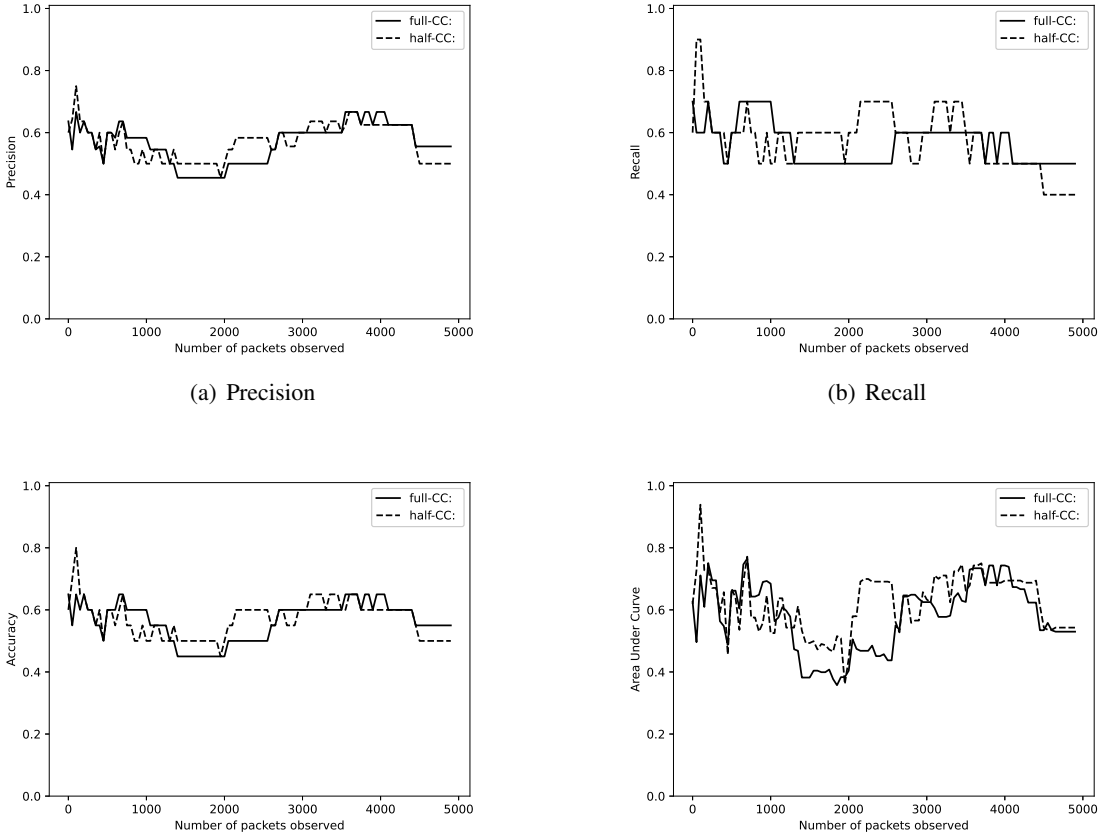
Figure 19: SHA2-384 worst alphabet: Statistical metrics according to the number of packets

evaluation scenario. Computational intensive CCs performing with the best alphabet may remain undetectable with our current implementation within the first 5,000 packets. The results for experiments utilizing the *worst* possible alphabet seem to achieve good to excellent detection rates, depending on the share of CC and the time consumption of the utilized hash-algorithm. Good detection rates can further be achieved after 1,000 packets, and for some implementations even after 500 packets.

However, the utilization of the best alphabet is not the standard scenario in the wild. The alphabet will be optimized under certain conditions, but will have to transmit information not previously known to CR and CS. I.e., the alphabet cannot generally be optimized. We conclude that our proposed detection approach is considerable to be applied for certain scenarios and even if it may not detect highly optimized alphabets with low CC share, it will force a limitation of information transferred by reducing either the share of CC information or by forcing the CS and CR to agree on a well-fitting alphabet in advance. Moreover, computational intensive CCs may be detectable by our approach if more than 5,000 packets are observed, which needs to be further researched.

4.5 Countermeasures and Limitations

Multiple potential countermeasures can be considered for our detection approach. The first countermeasure to mention is an inefficient CC implementation. Especially time-intensive operations besides the



(a) Precision

(b) Recall

(c) Accuracy

(d) Area Under Curve

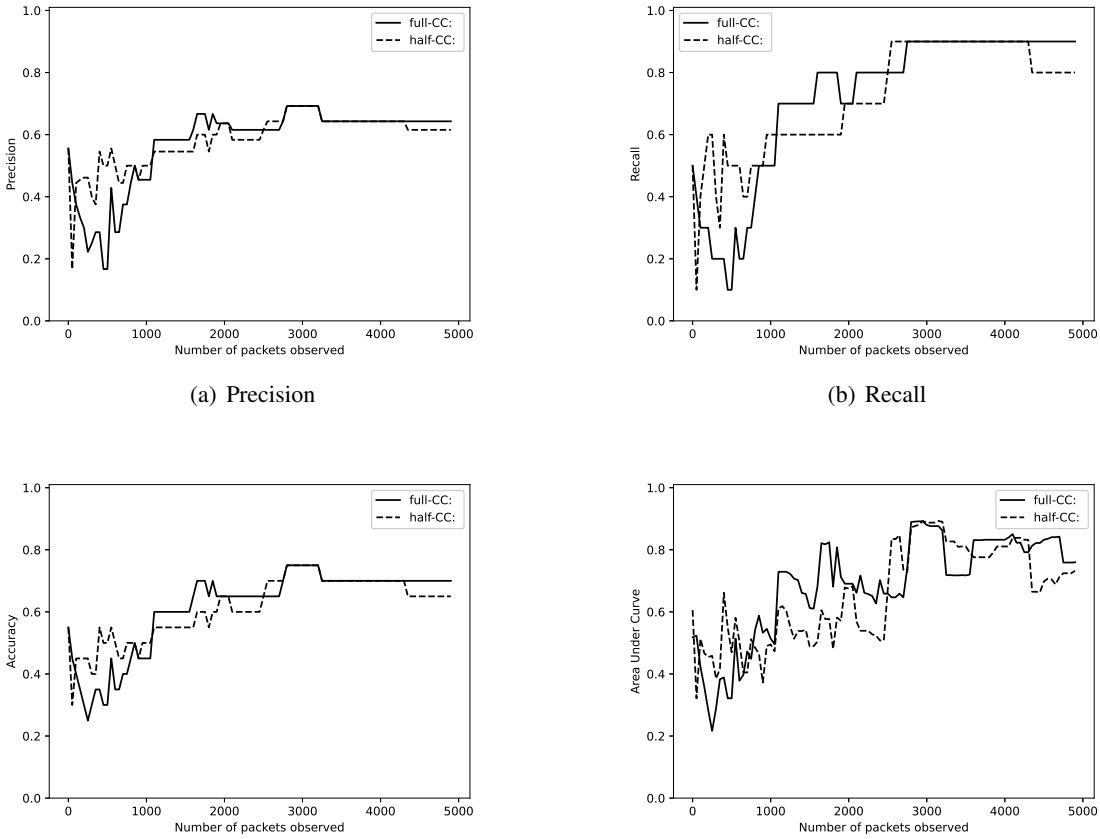
Figure 20: MD5 best alphabet: Statistical metrics according to the number of packets

hash-operations will lead to random delay and may prevent detection, e.g., read and write operations during the hash-reversion. Further, the creators of such a CC might add random sleep timers to obfuscate the appearance of malicious behavior or might try to normalize the packet runtime. These ideas were for instance described in [43, 40] and applied for traffic normalizers [44] to mitigate timing side and timing based covert channels, but they also can be applied to counter our detection approach. Further, the fewer the share of CC packets in a set of data, the less likely it is to detect the existence of covert information flows. Moreover, well-designed alphabets will decrease the chance of detection. Also, CRs with high computing capacity will be detected less likely because they can perform hash-operations more frequently. Especially systems, designed to perform hash operations, will decrease the detection rate as they will consume less time to perform those operations. Further, our detection approach might be limited to small setups with few hops between the observer and the CS. In other words, and given that all other variables are held constant, we can roughly assume that for our threshold-based detection heuristic, the following proportionality can be considered:

$$\text{Detectability} \propto \text{Number of Hops}^{-1}.$$

This can be reasoned by the following considerations:

- The time-consuming operations are performed by the CS. Placing the observer beyond this point will inhibit the detection.



(a) Precision

(b) Recall

(c) Accuracy

(d) Area Under Curve

Figure 21: MD5 worst alphabet: Statistical metrics according to the number of packets

- The more hops are passed between the observation point and the OR, the more the time difference will blur due to random time-consuming events (network jitter).
- The more hops are passed, the more likely it is for a packet to follow a different routing, leading to an altered runtime.

Thus, the longer the distance between the CS and the OR, the less significant the time difference observed will be. Though, it is crucial for our approach, that the warden can observe the flow *before* the computational intensive operation is performed. In our opinion, the best point to place the observer is directly in front of the CS, which might not be feasible in all practical scenarios.

5 Application Scenarios

Despite the mentioned countermeasures and limitations, our approach still can be applied within corporate networks to detect covert communication.

Obviously, a defender cannot always know the locations of a CS and a CR in a real environment. However, one might be able to locate potential CS and CR systems in a network. The three possible placements for an observer (further called warden) are presented in Figure 22. The warden in Figure 22(a) is placed between the OS and the CS and can observe the elongated packet runtimes of the repeated

computational intensive operations. However, since the intensive operations are performed by the CR, the result may be not as clear as the results of a warden placed in at the position of Figure 22(b). This warden has the best position as there are fewer systems hat can lead to random delays in between the warden, the CR and the OR. This may reduce events that can influence the runtime detection. The warden placed in Figure 22(c) will never be able to detect the CC as the computational intensive operation had already been performed by the CR and the runtime will not be observably influenced for this warden. Thus, the warden placement is crucial for this detection approach. If the warden is placed too close to the OS, the results might be falsified.

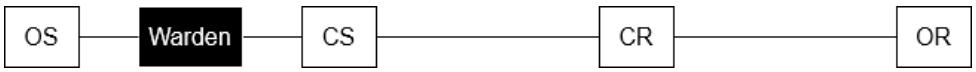
As already mentioned in section 4.5, the approach may be limited to small scenarios but still is a means to detect a CR in demilitarized zones (DMZ) and private network segments.

Our application scenario is as presented in Figure 23. Clients want to authenticate themselves to a web server that is hosted in a DMZ. The DMZ is encapsulated by an external and an internal firewall, and in addition secured by a proxy server. The web server itself requests a hash-check from an isolated authentication server in an internal network, the OR.

Figure 24 presents the location of the CS and the CR (red boxes) and the wardens (black boxes). We placed two CS and CR as examples, though there are more potential placements. Further, we introduced two wardens at positions that are crucial to the detection approach. The CS_1 is placed on the Internet and CS_2 is located in the DMZ, while the position of CR_1 is in the DMZ and CR_2 is placed in the internal network. The warden W_1 is installed in between the Internet and the external firewall, and the warden W_2 in between the DMZ and the internal firewall. CS_1 can transfer covert information to CR_1 (further referred to as **extCC-1**) and with CR_2 (**extCC-2**). CS_2 is able to communicate with CR_1 (**intCC-1**) and with CR_2 (**intCC-2**).

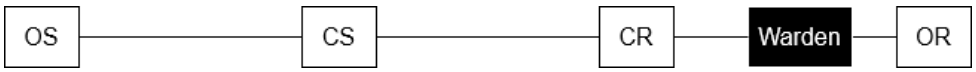
As the CC participants always take advantage of overt channel sub-routes (i.e., route fragments of the full route from OS to OR) to realize a MitM scenario, the actual placement of the CR can be determined in some cases. If only W_1 detects elongated runtimes, this indicates the CR is placed in between W_1 and W_2. If the CR had been placed between W_2 and the OR, W_2 would have also detected suspicious runtime behavior of request-reply cycles. In case of local OTP-authentications, the placements of the CS and the CR can be detected even more definite if packet flows are monitored on crucial points and collected by security instances like a security information and event management (SIEM).

As already mentioned, the wardens are placed at crucial positions, i.e., the transitions between each network segment. The warden W_1 will be able to detect modified hashes that a CS wants to submit to a CR in the secured network segments. The warden W_2 will be able to detect suspicious activities that may indicate a reversible CC based on computational intensive operations with a CR placed in the internal network. For the extCC-1, the warden W_1 is an equivalent to the scenario presented in Figure 22(b). It



(a) Warden placed between Overt Sender and Covert Sender

(b) Warden Placed between Covert Sender and Covert Receiver



(c) Warden Placed between Covert Receiver and Overt Receiver

Figure 22: Potential Warden Placements

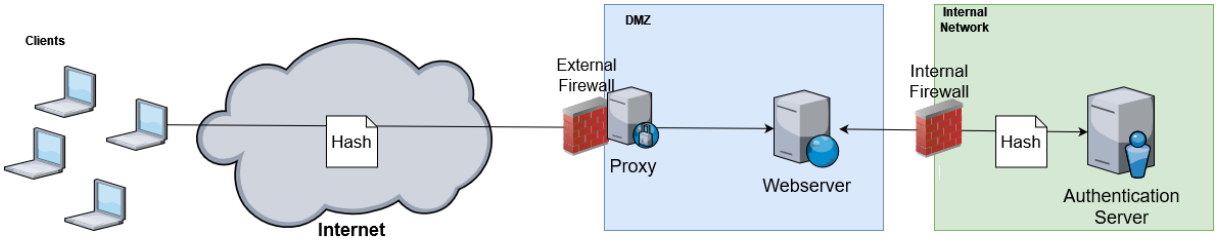


Figure 23: Application Setup

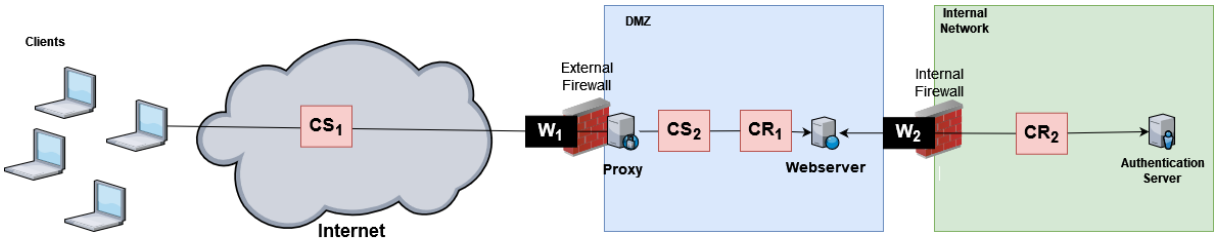


Figure 24: Placements of Wardens, Covert Senders and Covert Receivers

will be possible to observe extended packet runtimes, and there are few systems in between the warden and the CR. Further, extCC-2 will be observable for W_1 although the distance between the warden and the CR is longer than for the extCC-1; this is also the scenario presented in Figure 22(b). Despite the different placements of the CR, there will be no difference in the observable packet runtimes. This is explainable due to the fact that the same number of hosts will be passed from the warden to the OR (the authentication server). Both, intCC-1 and intCC-2 will also be observable by this warden. The packet runtime will be elongated because the computational intensive operations will be performed after the observation. This is the scenario presented in Figure 22(a). The warden W_1 is placed at a point to cover all possible CRs within a network, although the distance to the OR is the longest and results may be falsified due to this fact. The warden W_2 is placed at a point much nearer to the OR, so the passed nodes will be limited to a minimum. However, this warden will never neither be able to detect extCC-1 nor intCC-1 because the computational intensive operation has already been performed before the observation (see Figure 22(c)). It must be mentioned that this warden can be applied to all CCs addressing CR_2 and will also reach good detection results as the distance between the CR and the OR is short. This scenario corresponds to the warden placement presented in Figure 22(b).

As already described in section 4, the reference measurements are crucial for this detection approach. The wardens have to perform several experiments to enable high-quality results when located between the warden and the OR. In addition, a CC recognition can only be expected after 5,000 (or more) flows and depends on the utilized hash algorithm. Under certain circumstances, however, a reliable detection can be performed earlier like presented in Fig. 15 and Fig.19.

6 Conclusion

We demonstrated that the detection of a computational intensive reversible CC is feasible under certain conditions by observing packet runtimes. We investigated the influence of repeated SHA3-512, SHA3-256, SHA2-384 and MD5 hash operations on request-reply cycles of OTP authentications that are based upon hash chains. Our results give evidence that the utilized alphabet (i.e. the order of the symbols contained therein), as well as the probability of occurrence of each single symbol in the covert message has a high influence on the packet runtime of such computational intensive reversible CCs. For the worst alphabet our approach achieved good results for all tested algorithms. Especially the full-CC implementation of SHA2-384 had excellent detection rates. The performed experiments indicate that it is more likely to detect SHA3-based 512-bit hashes than SHA3-based 256-bit hashes or SHA2-384 and MD5-based implementations, regardless of the utilized alphabet. Further, we described countermeasures to prevent threshold-based packet runtime detection. Also, we discussed potential warden placements and their benefits and limitations as well as application scenarios in existing environments, showing that wardens must be carefully placed to expect usable detection results. In the future, we aim to improve our approach to achieve higher detection rates. To this end, we will evaluate additional methods to detect reversible CCs based upon computational intensive operations, especially operations consuming few resources. Furthermore, we plan to investigate the influence of the distance between the observation point, the CR, and the OR, and plan to compare different implementations (e.g., C and *libpcap* instead of Python and *scapy*).

References

[1] J. Keller and S. Wendzel. Reversible and plausibly deniable covert channels in one-time passwords based on hash chains. *Applied Sciences*, 11(2):731–741, January 2021.

[2] L. Caviglione, M. Gaggero, J. Lalande, W. Mazurczyk, and M. Urbanski. Seeing the unseen: Revealing mobile malware hidden communications via energy consumption and artificial intelligence. *IEEE Transactions on Information Forensics and Security*, 11(4):799–810, April 2016.

[3] T. Schmidbauer and S. Wendzel. Hunting shadows: Towards packet runtime-based detection of computational intensive reversible covert channels. In *Proc. of the 16th International Conference on Availability, Reliability and Security (ARES'21), Vienna Austria*, pages 1–10. ACM, August 2021.

[4] B. Preneel. Hash functions. In H.C.A.V. Tilborg and S. Jajodia, editors, *Encyclopedia of Cryptography and Security*. Springer US, 2011.

[5] Ralph C. Merkle. One way hash functions and DES. In *Proc. of the 9th Conference on the Theory and Application of Cryptology (CRYPTO'89), Santa Barbara, California, USA*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer, New York, August 1989.

[6] R.L. Rivest. The md5 message-digest algorithm. IETF RFC 1321, April 1992. https://rfc-editor.org/rfc/rfc1321.txt.

[7] T. Hansen and D.E.Eastlake 3rd. US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). IETF RFC 6234, May 2011. https://rfc-editor.org/rfc/rfc6234.txt.

[8] M. Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. Technical Report 202, Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, 2015.

[9] J. Liang and X. Lai. Improved collision attack on hash function md5. IACR Cryptology ePrint Archive, Report 2005/425, November 2005. `https://ia.cr/2005/425`.

[10] G. Bertoni, J. Daemen, M. Peeters, and G.V. Assche. Keccak. In *Proc of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece (EUROCRYPT'13)*, volume 7881 of *Lecture Notes in Computer Science*, pages 313–314. Springer Berlin Heidelberg, 2013.

[11] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, November 1981.

[12] C.H. Rowland. Covert channels in the TCP/IP protocol suite. *First Monday*, 2(5):1–1, May 1997.

[13] J. Cioranesco, H. Ferradi, R. Géraud, and D. Naccache. Process table covert channels: Exploitation and countermeasures. IACR Cryptology ePrint Archive, Report 2016/227, March 2016. `https://ia.cr/2016/227`.

[14] C. Chang and C. Lin. Reversible steganographic method using smvq approach based on declustering. *Information Sciences: an International Journal*, 177(8):1796–1805, April 2007.

[15] W. Mazurczyk, P. Szary, S. Wendzel, and L. Caviglione. Towards reversible storage network covert channels. In *Proc. of the 14th International Conference on Availability, Reliability and Security (ARES'19), Canterbury, UK*, pages 1–8. ACM, August 2019.

[16] V. Bindschaedler, R. Shokri, and C.A. Gunter. Plausible deniability for privacy-preserving data synthesis. *Proceedings of the VLDB Endowment*, 10(5):481–492, January 2017.

[17] A. Mileva and B. Panajotov. Covert channels in TCP/IP protocol stack - extended version-. *Open Computer Science*, 4(2):45–66, June 2014.

[18] S. Wendzel, S. Zander, B. Fechner, and C. Herdin. Pattern-based survey and categorization of network covert channel techniques. *ACM Computing Surveys*, 47(3):1–26, April 2015.

[19] S. Zander, G.J. Armitage, and P. Branch. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys & Tutorials*, 9(3):44–57, September 2007.

[20] T.B. Paiva, J. Navaridas, and R. Terada. Robust covert channels based on dram power consumption. In *Proc. of the 22nd International Conference on Information Security (ICS'19), New York City, USA*, volume 11723 of *Lecture Notes in Computer Science*, pages 319–338. Springer, Cham, September 2019.

[21] E. Zielińska, W. Mazurczyk, and K. Szczypiorski. Trends in steganography. *Communications of the ACM*, 57(3):86—-95, March 2014.

[22] K. Cabaj, L. Caviglione, W. Mazurczyk, S. Wendzel, A. Woodward, and S. Zander. The new threats of information hiding: The road ahead. *IT Professional*, 20(3):31–39, June 2018.

[23] P. Szary, W. Mazurczyk, S. Wendzel, and L. Caviglione. Design and performance evaluation of reversible network covert channels. In *Proc. of the 15th International Conference on Availability, Reliability and Security (ARES'20), Virtual Event Ireland*, pages 1–8. ACM, August 2020.

[24] J. Lim, H. Lee, S. Lee, and J. Kim. Invertible watermarking algorithm with detecting locations of malicious manipulation for biometric image authentication. In *Proc. of the 1st International Conference on Biometrics (ICB'06), Hong Kong, China*, volume 3832 of *Lecture Notes in Computer Science*, pages 763–769. Springer Berlin Heidelberg, January 2006.

[25] K. Yoo, M. Kim, and W. Lee. Invertible watermarking scheme for authentication and integrity. In *Proc. of the 6th Pacific-Rim conference on Advances in Multimedia Information Processing (PCM'05), Jeju Island, South Korea*, volume 3768 of *Lecture Notes in Computer Science*, pages 371–381. Springer, Berlin, Heidelberg, November 2005.

[26] C. Abad. Ip checksum covert channels and selected hash collision. Technical report, University of California, 2001.

[27] R.A. Kemmerer. A practical approach to identifying storage and timing channels: Twenty years later. In *Proc. of the 18th Annual Computer Security Applications Conference (ACSAC'02), Las Vegas, Nevada, USA*, pages 109–118. IEEE, December 2002.

[28] T. Shon, J. Seo, and J. Moon. A study on the covert channel detection of TCP/IP header using support vector machine. In *Proc. of the 5th International Conference Information and Communications Security (ICICS'03), Huhehaote, China*, volume 2836 of *Lecture Notes in Computer Science*, pages 313–324. Springer, Berlin, Heidelberg, October 2003.

[29] S. Cabuk, C.E. Brodley, and C. Shields. Ip covert channel detection. *ACM Transactions on Information and System Security*, 12(4):1–29, April 2009.

[30] S. Wendzel, F. Link, D. Eller, and W. Mazurczyk. Detection of size modulation covert channels using countermeasure variation. *Journal of Universal Computer Science*, 25(11):1396–1416, November 2019.

[31] J. Rahul, M. Sora, and L.D. Sharma. Dynamic thresholding based efficient qrs complex detection with low computational overhead. *Biomedical Signal Processing and Control*, 67:102519–102534, May 2021.

[32] A.M. Qureshi, N. Anjum, R.N.B. Rais, M.U. Rehman, and A. Qayyum. Detection of malicious consumer interest packet with dynamic threshold values. *PeerJ Computer Science*, 7:1–24, March 2021.

[33] M. Mushtaq, J. Bricq, M.K. Bhatti, A. Akram, V. Lapotre, G. Gogniat, and P. Benoit. WHISPER: A tool for run-time detection of side-channel attacks. *IEEE Access*, 8:83871–83900, April 2020.

[34] M.A. Ayub, S. Smith, and A. Siraj. A protocol independent approach in network covert channel detection. In *Proc.of the 22nd IEEE International Conference on Computational Science and Engineering (CSE'19), and IEEE International Conference on Embedded and Ubiquitous Computing (EUC'19), New York, USA*, pages 165–170. IEEE, August 2019.

[35] P. Nowakowski, P. Zórawski, K. Cabaj, and W. Mazurczyk. Network covert channels detection using data mining and hierarchical organisation of frequent sets: an initial study. In *Proc. of the 15th International Conference on Availability, Reliability and Security (ARES'20), Virtual Event Ireland*, pages 1–10. ACM, August 2020.

[36] A. Epishkina, M. Finoshin, K. Kogos, and A. Yazykova. Timing covert channels detection cases via machine learning. In *Proc. of the 8th European Intelligence and Security Informatics Conference (EISIC'19), Oulu, Finland*, pages 139–. IEEE, November 2019.

[37] I.G. Çavusoglu, H. Alemdar, and E. Onur. Covert channel detection using machine learning. In *Proc.of the 28th Signal Processing and Communications Applications Conference (SIU'20), Gaziantep, Turkey*, pages 1–4. IEEE, October 2020.

[38] M. Chourib. Detecting selected network covert channels using machine learning. In *Proc. of the 17th International Conference on High Performance Computing & Simulation (HPCS'19), Dublin, Ireland*, pages 582–588. IEEE, July 2019.

[39] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, and R. Baldoni. Android malware family classification based on resource consumption over time. In *Proc. of the 12th International Conference on Malicious and Unwanted Software (MALWARE'17), Fajardo, Puerto Rico, USA*, pages 31–38. IEEE, October 2017.

[40] S. Schinzel. *Unintentional and Hidden Information Leaks in Networked Software Applications*. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), April 2012.

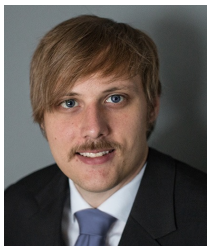
[41] I.E. Olatunji and C. Cheng. Dynamic threshold for resource tracking in observed scenes. In *Proc. of the 9th International Conference on Information, Intelligence, Systems and Applications (IISA'18), Zakynthos, Greece*, pages 1–6. IEEE, July 2018.

[42] C.Medlock and A.V. Oppenheim. Optimal roc curves from score variable threshold tests. In *Proc. of the 43rd International Conference on Acoustics, Speech and Signal Processing (ICASSP'19), Brighton, United Kingdom*, pages 5327–5330. IEEE, May 2019.

[43] J. Agat. Transforming out timing leaks. In *Proc. of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'00), Boston, Massachusetts, USA*, pages 40–53. ACM, January 2000.

[44] W. Mazurczyk, S. Wendzel, S. Zander, A. Houmansadr, and K. Szczypiorski. *Information Hiding in Communication Networks. Fundamentals, Mechanisms, and Applications*. Wiley-IEEE Press, 2016.

Author Biography



Tobias Schmidbauer is a PhD student at Fernuniversität in Hagen, Germany. Before, he studied computer science in public management (Verwaltungsinformatik) at Hochschule Hof / Hochschule für den öffentlichen Dienst Hof, Germany and received his diploma in 2016. Thereafter, he started studying practical computer science in part-time at Fernuniversität in Hagen, Germany, and graduated with an M.Sc. degree in 2019. Alongside his studies, he worked from 2007 until 2017 for the datacenter of the tax administration of the German federal state of Bavaria. Since 2017, he works full-time for the Bavarian State Office for Information Security as a penetration tester. In addition, he was deputy information security officer of the Bavarian State Office for Information Security until the end of 2020 and has held this position in full-time since the beginning of 2021.

Steffen Wendzel is a professor of information security and computer networks at Hochschule Worms, Germany, where he is also the scientific director of the Center for Technology and Transfer (ZTT). In addition, he is a lecturer at the Faculty of Mathematics & Computer Science at the FernUniversität in Hagen, Germany, from which he also received his Ph.D. (Dr. rer. nat., 2013) and Habilitation (2020). Before joining Hochschule Worms, he led a smart building security research team at Fraunhofer FKIE in Bonn, Germany. Steffen (co-)authored more than 170 publications and (co-)organized several conferences and workshops (e.g. IWSMR'19–'22, EICC'20, GI Sicherheit'16) and special issues for major journals, such as IEEE Security & Privacy (S&P), Elsevier Future Generation Computer Systems (FGCS), Journal of Universal Computer Science (J.UCS), Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), and IEEE Transactions Industrial Informatics (TII). He is editorial board member of J.UCS and Journal of Cybersecurity & Mobility (JCSM). His website: `https://www.wendzel.de`.