# Remote IP Protection using Timing Channels

Ariano-Tim Donda[1,2], Peter Samarin[1,2], Jacek Samotyja[1],
Kerstin Lemke-Rust[1], and Christof Paar[2]

[1]*Bonn-Rhein-Sieg University of Applied Sciences, Germany*
[2]*Ruhr-Universität Bochum, Germany*

4 Nov 2014

We introduce the use of timing channels for digital watermarking of embedded hardware and software components. In addition to previous side channel watermarking schemes, timing analysis offers new perspectives for a remote verification of mobile and embedded products. Timing channels make it possible to detect the presence of a watermark solely by measuring program execution times.

We propose schemes for embedding authorship and fingerprint marks that are built upon conditional timing delays. We provide experimental evidence by protecting an implementation of an image binarization circuit on an FPGA board that is connected over Ethernet to a remote PC. The circuit constantly leaks the watermark over the timing channel by modulating its execution time, which is successfully detected by using an oscilloscope and an EM probe, as well as by using software on a remote PC. Our solution for a remote verification is of special interest for highly performant services as they force an adaptive adversary towards enhanced costs in time, memory, and circuitry when bypassing these schemes.

**Keywords:** IP Protection, Digital Watermarking, Timing Channel, Timing Analysis, Side-Channel Analysis, Authorship Watermark, Fingerprint Watermark, FPGA Implementation, Embedded Systems.

ariano-tim.donda@rub.de
peter.samarin@h-brs.de
jacek.samotyja@h-brs.de
kerstin.lemke-rust@h-brs.de
christof.paar@rub.de

## 1 Introduction

Digital watermarking schemes look back on a long tradition. Basically, a watermark is an identifying information that is embedded in media. It has to fulfill three requirements. First, the watermark shall not impede normal use of the watermarked media, second, the watermark shall become verifiable if it undergoes a specific test, e.g., the paper watermark of a banknote becomes visible in case of exposure to light (Cox et al., 2008), and third, an unauthorized party should not be able to remove or alter the watermark.

Today, there is a variety of digital watermarking schemes invented for audio, images, video, and software media for many purposes. Due to this diversity and depending on the purpose, different properties of digital watermarking schemes are important. If it shall be a hard problem to remove a watermark from a media, robustness is needed. Otherwise, if authenticity of a media shall be guaranteed, this calls for fragility of the watermark in case of any modification of the media. In (Nagra et al., 2002) a valuable taxonomy on digital watermarks can be found.

Our security objective is to protect embedded systems against plagiarism. In this paper we introduce watermarks based on conditional timing delays that are deeply embedded into software and hardware components. A watermark becomes verifiable if timing differences such as program execution times or parts thereof are analyzed. Timing analysis for detecting digital watermarks can be done by measuring power consumption or electromagnetic (EM) radiation of a device (Kocher et al., 1999; Mangard et al., 2007), and even remotely without the need for any special equipment.

This paper provides high-level schemes for embedding an authorship mark and a fingerprint mark (Nagra et al., 2002) in the timing side channel in order to protect embedded hardware and software. The schemes and their realizations in the timing channel are presented in Section 3. An authorship mark embeds information identifying its author. A fingerprinting mark embeds information identifying the serial number of the purchaser of the component. For both marks, robustness is an important security property.

In Section 4, we provide evidence of a successful implementation of these schemes in an image binarization circuit on an FPGA that is connected over Ethernet to a PC. For timing analysis, we tested three different measurement settings: (i) in proximity to the FPGA board using an EM probe, (ii) at the Ethernet cable using a contact-based measurement and an EM probe, and (iii) on a remote PC. The first two settings use an USB oscilloscope, while the third set-up uses only an open-source software library for capturing UDP packets on the PC.

Electromagnetic radiation and power consumption side channels have been used to leak a watermark before (Becker et al., 2011, 2010), however, previous work did not consider the timing side channel. The advantage of using the timing side channel is that the watermark can additionally be verified at a remote network device. Exploitation of the timing side channel has been explored by (Bernstein, 2005; Page, 2002) to infer the secret key of a remote server,

however, not to transmit a watermark, as in our approach. Section 2 views our approach in context of other work done in this area.

## 2 Related Work

The first known use of a timing channel traces back to inter-process communication on a secure operating system using dynamically shared resources in order to bypass information flow models. Such a timing channel can be activated by page faults, CPU demand, segment activation, disk cache loading, and other means (Vleck, 1990).

Timing analysis was also the first published side channel based attack (Kocher, 1996) that provides a methodology to compromise keys of RSA, DSS and other cryptosystems by measuring the execution time of the overall cryptographic operation. For success, it is required that the execution times of the elementary operations of a modular exponentiation are data dependent. As the secret key is the exponent, successively finding out the sequence of elementary operations reveals the secret key. More recent side channel attacks exploit timing delays on CPUs such as cache attacks (Bernstein, 2005; Page, 2002) and branch prediction (Aciicmez et al., 2006).

Timing measurements are also of high interest for traffic analysis of anonymizing networks such as Tor (The Onion Router), e.g., cf. (Murdoch and Danezis, 2005; Wang et al., 2005). It was shown that low latency anonymizing networks are susceptible to timing attacks that actively add timing delays to selected packets. In (Wang et al., 2005) watermarking of packets of a Skype call was done by actively imprinting time delays on packets according to a 24-bit watermark on one communication endpoint and it was shown that the watermark can be revealed after passing through the anonymizing network on the other endpoint.

The idea of combining side-channel analysis and digital watermarking for protecting intellectual property (IP) was first developed in (Becker et al., 2011, 2010). In the earlier work (Becker et al., 2010), the authors introduced side-channel watermarks for integrated circuits. In the later work (Becker et al., 2011), an implementation in embedded software was suggested. Their contributions are based on power analysis as introduced by (Kocher et al., 1999). Power analysis requires tapping a power pin of the device under test and measuring it with an oscilloscope. In (Becker et al., 2010) the authors present a spread spectrum watermark and an input-modulated watermark. The spread spectrum watermark amplifies the output bit of a pseudo random number generator (PRNG) or, alternatively, a stream cipher with a leakage circuit in each clock cycle. Verification is done by simulating the outcome of the PRNG and correlating it with the power measurements of the leakage. The input-modulated watermark uses a combinatorial function of some input bits that computes one output bit that is sent to the leakage circuit. Verification is done by correlation power analysis. Basically the same scheme is proposed in (Becker et al., 2011) for embedded software. Herein, it is made more concrete: the authors use a combination function of 32-bit input bits and a 64-bit watermark key to compute

one bit that is leaked out over the power consumption.

## 3 Watermarking through Timing Channels

### 3.1 The Adversary Model

In our model, the owner of IP rights aims to protect an embedded product against unauthorized use. For this purpose, the product is watermarked in order to detect fraud due to unauthorized use of copies, plagiarism, and their distribution chain. The owner of the IP rights co-operates with the watermarker $\mathcal{W}$ who embeds digital watermarks based on conditional timing delays, compiles the sources, and distributes the product under copyright.

In this paper we use the term $f$ to denote a function that has been protected by a watermark. Let $f'$ denote a possibly different function with a functionality that is similar to $f$. Function $f'$ is the object of investigation done by verifier $\mathcal{V}$ in order to decide whether it contains the watermark of $\mathcal{W}$ or not. The function $f$ is assumed to have a data input and a data output channel. Both may be optional under certain conditions that are detailed further in this paper.

Our security objective is twofold:

$O_1$: Verifier $\mathcal{V}$ detects copyright violations of function $f$.

$O_2$: Verifier $\mathcal{V}$ discloses the distribution chain of the illegal copy of $f$ in order to identify the issuer of the illegal copy or plagiarism.

In our model, the adversary is in possession of the compiled machine code of $f$. The adversary transforms the binary code: $f \rightarrow f'$ and distributes $f'$, eventually as part of another program. Transformations include subtractive, distortive, and additive attacks to the embedded watermark.

In this work, we act on the assumption that a complete reverse engineering of $f$ is a hard problem. Regarding FPGAs this is a reasonable assumption as decoding tools for FPGA bitstreams are not publicly available. Software reverse engineering requires disassembling and debugging tools which are available for many processors, but it is manual work and very time consuming. The resistance of a software implementation against reverse engineering can be significantly enhanced by using anti-debugging and anti-disassembly techniques, obfuscation, secret splitting, and encryption of parts of program code that is decrypted at run-time (Aycock, 2006).

### 3.2 The Timing Channel

The timing channel is realized by using a start and end time of the regular input and output channel of function $f$. The timing channel can be built on any time difference between two successive observable events within $f$ provided that they can be measured, e.g. network activities of $f$ or special characteristics in a power or EM trace. Hereby, we assume that the start

time can be triggered or observed and the end time can be observed. In order to send data over the timing channel, the sender introduces timing delays into the regular output channel and the receiver can read the data from the timing channel by noting the time difference between input and output.

In practice, the task of the receiver is a signal detection problem. The time difference is considered as a physical observable that implicitly depends on the conditional timing delay besides other deterministic contributions and noise. A measurement outcome of this observable is denoted by $\Delta_t$. Its origin is one of two or more populations that are labeled with different timing delays. The receiver decides for the population yielding maximum likelihood and decodes the information on the timing channel accordingly.

### 3.2.1 Binary Method

The simplest implementation of the timing channel constitutes a binary method, i.e., there are two populations and a zero bit is encoded without delay and a delay $\delta_t$ is added if the bit is one. Assuming that both events are equally likely the following decision rule applies: If $\Delta_t$ is longer than the mean time difference $\overline{\Delta_t}$, the output bit $b$ on the physical output channel is decoded to bit one, otherwise it decodes to bit zero.

$$b = \text{decode}(\Delta_t) := \begin{cases} 1 & \text{if } \Delta_t \geq \overline{\Delta_t} \\ 0 & \text{if } \Delta_t < \overline{\Delta_t} \end{cases}$$

### 3.2.2 Sliding Window Method

An alternative implementation of the timing channel is a sliding window method that can output more than one bit in each $\Delta_t$. More precisely, each zero bit is separately encoded without any timing delay and an $l$-bit sequence of ones is encoded with a delay of $l\delta_t$, i.e. $\Delta_t$ is proportional to the number of ones in a run given that $l$ is smaller than an implementation specific maximum run length $m$. The output bits $w$ on the physical output channel are decoded by empirical statistics using probability distributions $P_{i,j,m}$ with $i \in \{0,1\}$, $1 \leq j \leq m$ if $i = 1$ (i.e. a run of bit one) and $j = 1$ if $i = 0$ (i.e. a zero bit). In the simplest case decoding can be done by computing the differences of $\Delta_t$ from the means $\mu_{i,j,m}$ of all probability distributions and deciding for that probability distribution that minimizes the difference:

$$(b,l) = \text{decode}(\Delta_t) := \min \arg_{i\in\{0,1\},1\leq j\leq m} |\Delta_t - \mu_{i,j,m}|$$

If $l$ is incorrectly decoded, the sliding window method leads to a de-synchronization of sender and receiver and follow-up errors. Because of that, a backtracking algorithm needs to be foreseen to correct such wrong decodings. Alternatively, a reset function of $f$ may be used

for re-synchronization.

From perspective of performance, it is desirable to not substantially reduce the overall performance by introducing the timing delay. This requires that the computation of $f$ is decoupled from its output, so that the application does not remain idle while waiting for the delay to run out. This can be accomplished by computing outputs during a delay and storing them in a buffer, so that when the delay is finished, the next output can be sent immediately.

#### 3.2.3 Notation

In the following, we generalize the timing channel by using functions sndTC and rcvTC to indicate the data transmitted and received over the physical channel. $l$=sndTC$(c,i)$ encodes the delay corresponding to bitstream $c$ starting at offset $i$ and outputs the number of bits $l$ that are sent. $(b,l) =$ rcvTC$(\Delta_t)$ decodes an $l$-bit run of bit $b$ on the receiver side. $w$ denotes a run with bit length $l$ i.e. $w = (b,l)$. The auxiliary function cmp$(c,i,w)$ checks whether the bits starting at offset $i$ in $c$ are equal to $l$-bit run $w$ and outputs 'true' or 'false'.

### 3.3 Authorship Watermarks

In order to detect copyright violations of function $f$, i.e., our security objective $O_1$, we propose the use of authorship watermarks. Authorship watermarks are used to identify the owner of IP. We introduce two authorship watermarking schemes: a codeword scheme and a challenge-response scheme.

#### 3.3.1 Codeword Scheme

The codeword scheme cyclically broadcasts a fixed secret $n$-bit codeword $c_{CW}$ on the timing channel. This scheme does not require any input data channel. Fig. 1 shows the protocol for verifying one or a few bits of the codeword. $\mathcal{V}$ checks in each protocol run, whether the decoded bit of the measured execution time corresponds to the expected bit of the codeword. The number of successful authentications *suc* is counted. Both parties continue to increment the offset $i$ in the codeword for further function calls to $f$. An extension of this watermarking scheme towards an output sequence of a linear feedback shift register (LFSR) instead of a codeword is an alternative protocol design.

#### 3.3.2 Challenge-Response Scheme

This proposal is based on a common cryptographic challenge-response scheme (cf. (Boyd, 2003)). The security aim is that $f$ authenticates to verifier $\mathcal{V}$. This scheme introduces a timing delay depending on the outcome of an encryption algorithm $E$ that is parameterized with a

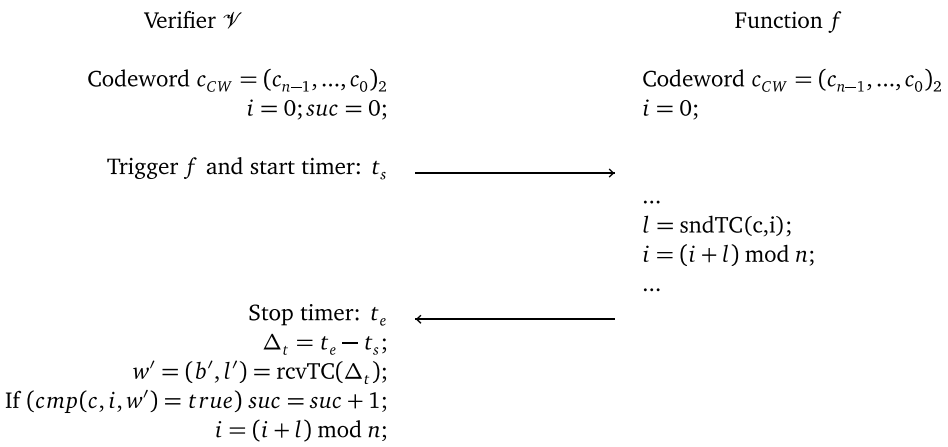



Figure 1: Protocol for embedded authorship watermark with a codeword $c_{CW}$.

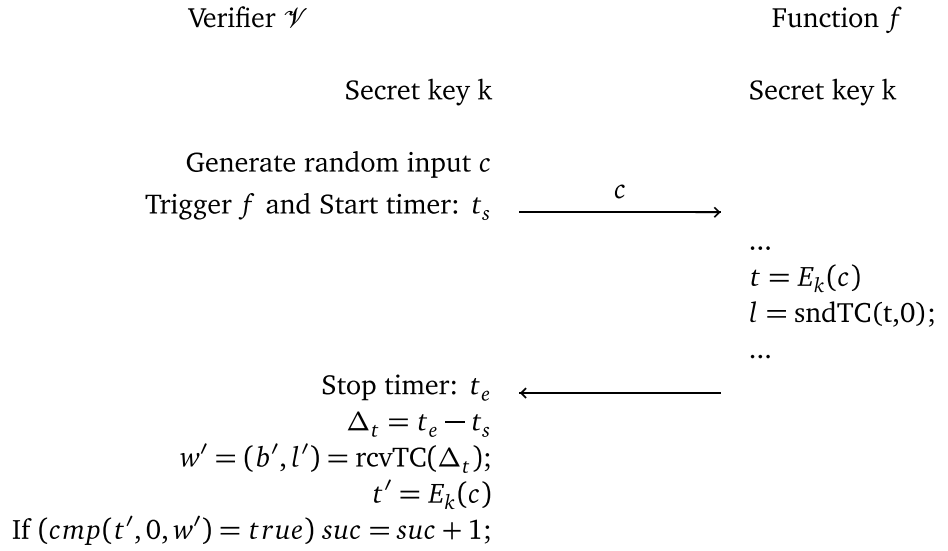



Figure 2: Protocol for embedded challenge-response authorship watermark using a block cipher $E_k$.

secret key $k$ and initialized with data $c$ that are input to function $f$. The key is the authorship mark. Therefore, it requires a data input channel for receiving the challenge of $\mathcal{V}$.

At each execution of $f$, the input $c$ is taken as an input to cipher $E$. If the cipher is a block cipher, function $f$ computes $E_k(c)$. If the cipher is a stream cipher, then $E$ is initialized with a secret key $k$ and the initialization vector (IV) $c$. $E_k(c)$ is then transmitted to $\mathcal{V}$ over the timing channel. Verifier $\mathcal{V}$ knows the cipher and its secret key and is able to check the correctness of the result for every input data $c$. The protocol for verifying one bit is given in Fig. 2.

The protocol design can be optimized for performance if the number of ciphering operations

can be reduced for successive invocations to $f$. This can be achieved with a stream cipher in which the keystream is initialized at the first invocation of $f$ using data input $c$. Starting with offset 0 of the key stream bits, at further invocations of $f$ the pointer to the key stream is incremented by the number of sent bits until a reset occurs. A similar but limited optimization is possible for block ciphers where successively all bits of an output block are sent on the timing channel before the encryption is executed again.

### 3.4 Fingerprint Watermarks

Fingerprint marks are intended to be invisible. For this proposal we re-visit the idea of an Easter egg watermark (Collberg and Thomborson, 1999). An Easter egg watermark performs some action if it receives a highly unusual input from the user. This action is assumed to be definitively detectable by the user.

For an Easter egg, timing delays do not slow down the performance of function $f$ in normal use so that a long delay on the timing channel is feasible. We assume that $\mathcal{V}$ possesses a list of secret keys that are allocated by $\mathcal{W}$. In case that the number of distributors and therefore the number of fingerprints is very high, our protocols can be extended with a tree search to speed up the verification process.

Our aim of a challenge-response scheme is to use cryptographic means in order to insert the timing delay only if the verifier $\mathcal{V}$ has been successfully authenticated before. Therefore, we use a successful challenge-response authentication as trigger for the visibility of the watermark. If the authentication fails, the watermark remains invisible.

For this scheme shown in Fig. 3, it is necessary that the data input channel and data output channel are available and can be used for the watermarking scheme. The fingerprint watermark protocol requires two runs of $f$. $E$ is a secure encryption algorithm that is parameterized with a secret key $k$, which is the fingerprint of the distributor in this scheme. The challenge-response protocol runs as follows. In each protocol run, function $f$ generates a new random output value $r$, computes $t = E_k(r)$ and stores the result. Verifier $\mathcal{V}$ obtains $r$ from the data output channel, selects one secret key $k'$ from its list of secret keys, computes $c = E_{k'}(r)$, and sends $c$ on the data input channel to function $f$. If $c = t$ holds, function $f$ causes a long timing delay that signals a successful authentication to $\mathcal{V}$. Thereby, $\mathcal{V}$ reveals the fingerprint key that was originally built in function $f$.

### 3.5 Security Analysis and Implementation Considerations

The adversary can try to remove, rearrange and add code parts to $f$.

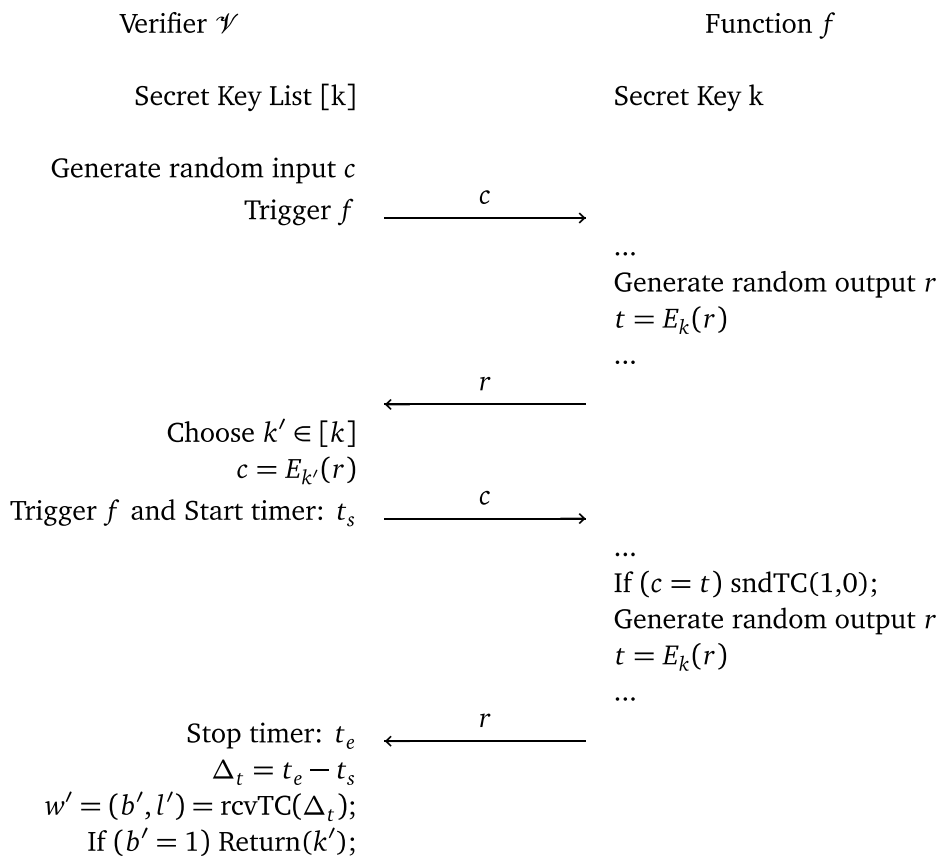



Figure 3: Protocol for embedded challenge-response fingerprint watermark using a block cipher $E_k$.

#### 3.5.1 Subtraction

The challenge of subtraction is to identify single parts of the binary code containing the timing watermark and to leave the main part of $f$ intact. Identification of such parts may be feasible with reverse engineering, possibly with the help of side channel analysis. Any measure that enhances the robustness of the implementation helps in resistance against subtraction. Further, start-up tests such as known-answer tests for encryption units and timing channel encoding help to detect subtractions. Subtraction attacks are considered to be a relevant threat for software, but much less for hardware implementations.

#### 3.5.2 Distortion

The challenge is to reorder the code of $f$ in order to destroy the timing delay but to leave the main part of $f$ intact. Again, such an attack strongly depends on reverse engineering results and is considered to threaten software and much less hardware.

#### 3.5.3 Addition

The addition attack aims to hide the timing watermark. If the adversary adds random delays this enhances the noise level of the timing channel and the number of queries a remote verifier has to ask to gain a specific confidence level, but it cannot hide the watermark.

One strong addition attack is obvious: If the execution time of $f'$ is set to a constant value the timing channel is blocked for remote verifiers. This addition can be pre-programmed for the codeword scheme and needs to be adjusted in real-time for the challenge-response schemes. The drawback of this attack is that such a wrapper attack slows down overall performance of $f'$ as the constant time difference needs to be set to the maximum time difference of $f$ that can occur. Because of this, the sliding window method is advantageous in pushing the execution time of $f'$ to significantly higher limits which may make such a wrapper attack inefficient for an adversary. Further, the fingerprint mark is rarely affected as the timing delay is long. Besides performance penalty a wrapper attack requires sufficient memory for intermediate storage and possibly additional circuits, thereby imposing additional costs to the adversary. For embedded devices with significant I/O load and tight timing requirements this wrapper attack may lead to data loss. Note that local verifiers using side channel analysis may be able to detect the presence of a wrapper attack.

## 4 Experimental Results with an FPGA Implementation

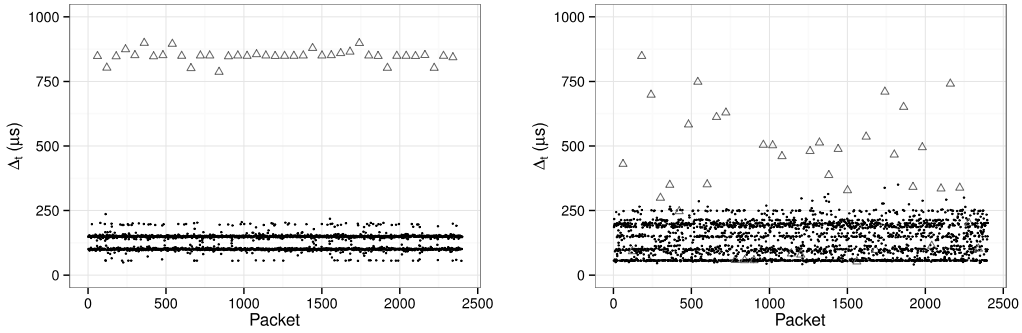
To demonstrate the feasibility of timing watermarks, the protection scheme has been applied to a simple computer vision task of image binarization implemented on an Altera DE2-70 board. The board has an FPGA with 70k logic elements that is based on SRAM and needs to be reconfigured with the bitstream at each power-up.

### 4.1 Image Binarization Circuit

The image binarization circuit converts images with 8 bits per pixel into images with 1 bit per pixel, thus, each pixel is either dark or bright. The decision to convert the pixel to dark or bright pixel is made by comparing the 8-bit pixel values to a user-controlled threshold. The images are obtained from a camera running at 119 frames per second with a resolution of $640 \times 480$ pixels. The binarized images are split into 60 parts and sent over an Ethernet interface to a fixed IP address using the UDP protocol. Each UDP packet is numbered from 0 to 59 to make it possible for the receiver to properly reassemble the images. Binary images are reconstructed using the packet number and the pixel data.

### 4.2 Establishing the Timing Channel

We use the binary method to establish a timing channel between the board and a generic receiver that can be an EM probe, a power consumption probe, or a remote PC listening on the network. The timing channel is realized by introducing artificial delays into the packet transmitting circuit on the FPGA board. Thus, presence of a delay denotes a binary "1", and the lack thereof means a binary "0", cf. Sect. 3.2 for the use of the binary method. Since each frame consists of 60 packets, it is possible to send 60 bits of information per image over the timing channel.




(a) Delay = 0 $\mu s$ (b) Delay = 120 $\mu s$

Figure 4: Time differences between consequent packets captured on the PC for different delays. Gray triangles show the timing differences between the first packet of image frame and the last packet of the previous image frame. Black dots show the time differences between packets in the same frame.

The data is captured on the PC by using the libpcap library, which provides elementary procedures for capturing and analyzing network packets. For every two consecutive packets, the time difference is computed by subtracting their respective timestamps. Fig. 4 shows the timing channel of a challenge response authorship watermarking scheme of 2400 UDP packets obtained from 20 image frames. Time differences between the last packet of one frame and the first packet of the next frame, as denoted by gray triangles, are much larger than the time differences between packets of the same frame. When no delay is introduced in the timing channel, there is little variance in the time differences. In contrast, a large delay results in high variances in the timing channel. If many delays are introduced in one frame, the time difference between the last frame and the subsequent frame is shorter.

### 4.3 Circuit Watermarking

The binarization circuit is protected by two proposed authorship watermarking schemes.

#### 4.3.1 Codeword Authorship Watermark

A 60-bit codeword watermark was implemented by using a 60 bits shift register that has a 1-bit output and wraps around upon each shift operation. Again, 60 bits are chosen for convenience, because exactly 60 packets are necessary to transmit one frame. The design overhead measured by Altera's Quartus IDE for this watermarking scheme is 60 logic cells.

Before sending a packet, the output bit of the shift register is consulted. If its value is a binary "1", a delay is introduced, otherwise the packet is sent right away. After sending a packet, the shift register is shifted, and its output is set to the next bit in the bitstring. In this way, a codeword authorship watermark is repeatedly transmitted over the timing channel.

#### 4.3.2 Challenge-Response Authorship Watermark

In the challenge-response authorship watermark the fixed codeword is replaced with the Trivium (Cannière, 2006) stream cipher with a fixed key and an input-dependent initialization vector (IV). We chose Trivium because it is well-suited for hardware implementation and it has a simple design. Trivium operates with an 80-bit initialization vector and an 80-bit key. In our system, the IV for the Trivium circuit is obtained from the first 80 binarized pixels of each frame. The design overhead measured by Altera's Quartus IDE for this watermarking scheme is 320 logic cells.

After obtaining the IV, which takes 80 clock cycles, the internal state of Trivium is initialized during 1152 clock cycles. Thus, the first bit of the stream cipher is available after 1232 clock cycles after receiving the first image pixel. However, it will not be used before processing 8 image lines, which takes at least $640 \cdot 8 = 5120$ clock cycles. After sending 60 bits on the timing channel, the Trivium circuit is reset and initialized again. This has the advantage that the verifier can analyze the timing channel already after receiving a single frame.

### 4.4 Timing Analysis for Watermark Recognition

#### 4.4.1 EM Emanation in Proximity of the FPGA Board

Timing measurements were done (i) in proximity of the FPGA board using an EM probe, (ii) at the Ethernet cable using a contact-based measurement and an EM probe, and (iii) on the remote PC by using the libpcap library. The first two settings used the USB oscilloscope Picoscope 5203 by Picotech while the third set-up uses only the libpcap library on the PC. Our objective is to reveal the timing delay from the measurements without the help of any special triggers from the FPGA board. For illustration purposes, such a trigger indicating the presence of a 10 $\mu$s delay is plotted in Fig. 6(a).

This measurement set-up corresponds to a standard EM set-up for side-channel analysis cf. (Mangard et al., 2007). The positioning of the EM probe RF-U5-2 by Langer EMV is shown

in Fig. 5(a). The sampling rate was 1 GHz. Fig. 5(b) includes two single measurements, the measurement in the top does not include the delay, whereas the delay is present in the bottom trace. We added a vertical line in Fig. 5(b) to roughly indicate the relevant pattern used for the detection. The time difference is about 40 ns which corresponds to two clock cycles. As a result, the minimum delay of two clock cycles can be reliably detected so that one single measurement trace is sufficient to read-out more than 60 bits that are transmitted over the timing channel.

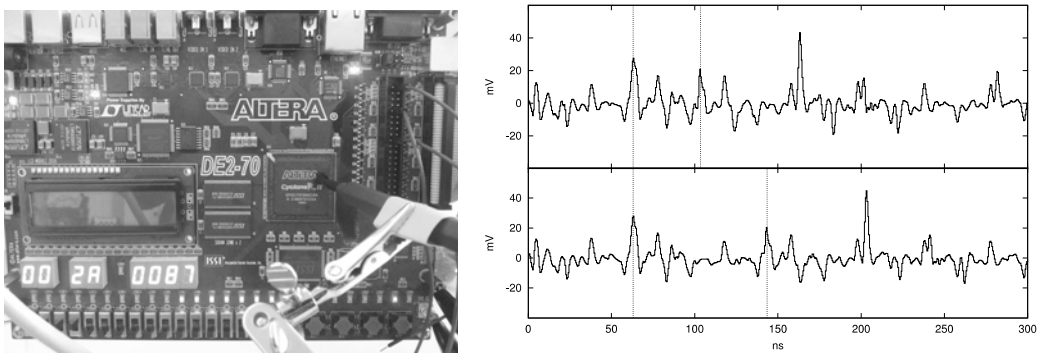
#### 4.4.2 Timing Analysis at the Ethernet Cable

In this experiment it is our aim to study to which extent timing analysis can be conducted at the Ethernet connection between the FPGA and the remote PC. In the first experiment, we cut off and directly contacted the Ethernet cable with a passive probe at a sampling rate of 1GHz. Fig. 6(b) shows that the minimum delay of two clock cycles is clearly visible in the power trace. We note that reducing the sampling rate is feasible down to about 25 MHz where it still yields a sufficient precision for the timing measurement.

In an alternative experiment, we positioned a near-field probe near the Ethernet cable. Also, these experiments were successful with a precision of down to the minimum delay of two clock cycles. Fig. 6(c) shows timing measurements for the EM probe.

#### 4.4.3 Remote Verification Using a PC

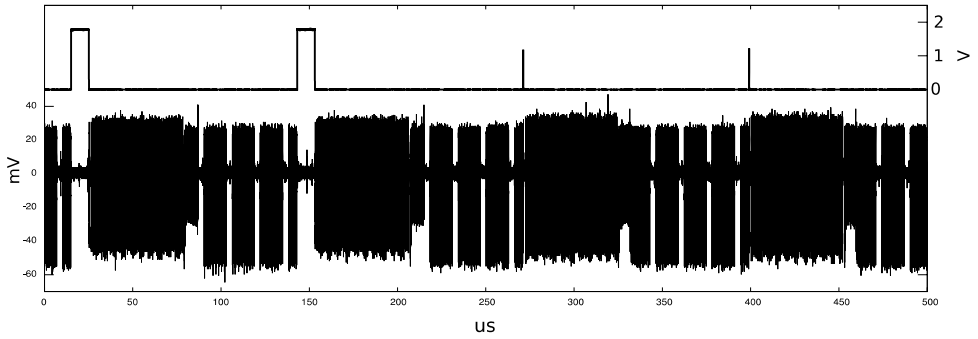
The remote verification was done on a PC with a 3.10 GHz quad-core Intel Xeon E3-1220 processor running 64-bit Debian with kernel 3.2.0-4. During remote verification, the presence of an authorship watermark is confirmed by connecting the FPGA board over its regular data channel to the PC. This has the advantage that no additional equipment is needed. The FPGA



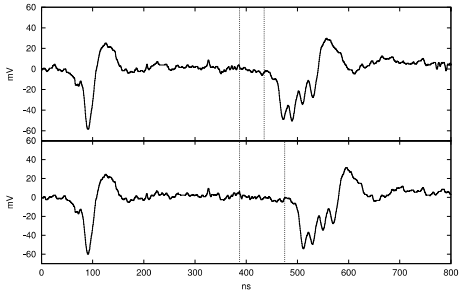

(a) Measurement set-up for the EM emanation in the near field.

(b) Timing analysis at the EM setup. The timing difference is clearly visible in single EM traces

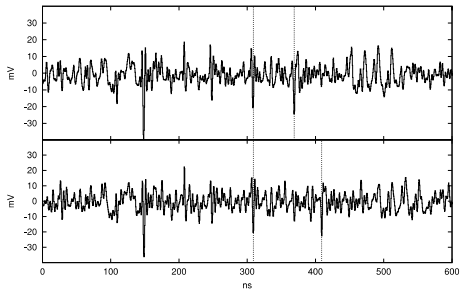
Figure 5: Results of the EM-Setup in proximity of the FPGA board.




(a) Probe connected to the Ethernet cable and special trigger signal for a delay of 10 $\mu s$.




(b) Probe connected to the Ethernet cable.




(c) EM Probe near the Ethernet cable

Figure 6: Results at the Ethernet connection.

board and the PC are positioned in two different rooms and are connected to the department network that is used by approximately 50 people. Altogether, there are two routers and three switches that separate the board from the PC. Unlike the measurements done on a USB oscilloscope, remote verification introduces the problem that the libpcap library assigns a timestamp to a packet at the moment when it is transferred from kernel space to user space, and not when it is received by the Ethernet controller of the PC.

In order to find out the dependency between the delay in the timing channel and the confidence of the verifier, several measurements with different delays in the timing channel have been carried out. The delays varied from 0 to 120 $\mu s$, with the step size of 20 $\mu s$ which resulted in 6 datasets.

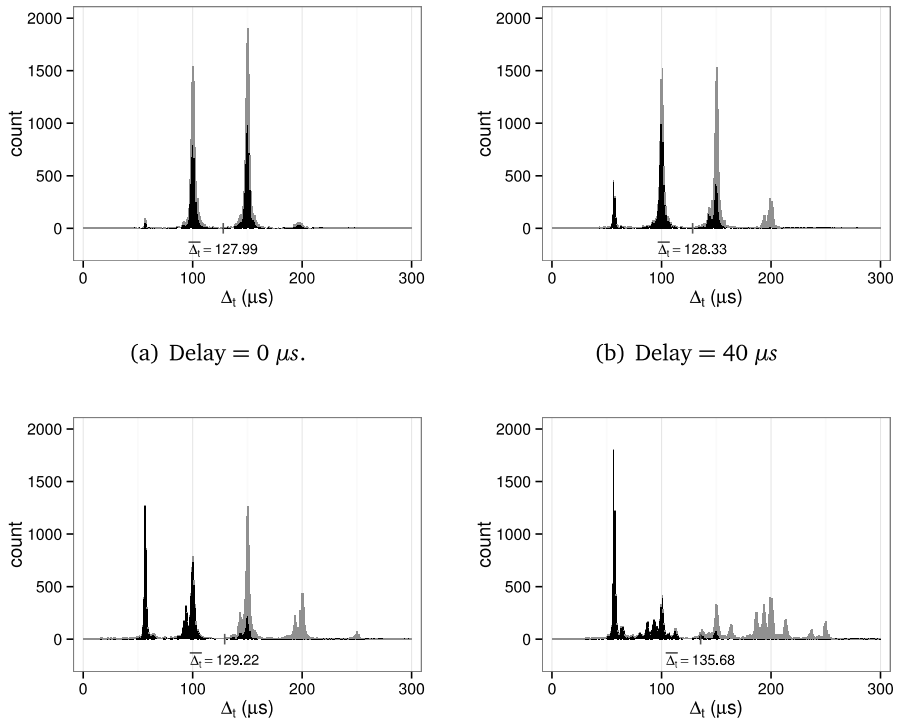
The performance of this approach has been evaluated on the captured data by computing the ratio between the incorrectly recognized timing channel bits and the total number of bits sent over the timing channel. Fig. 7 shows the distribution of the time differences between the packets of four datasets with different delays. When no additional delay is introduced, no data is sent over the timing channel because the packets are indistinguishable from each other.[1]

[1]The two peaks in absence of delay in Fig. 7 (a) arise from the combination of operating system and the kernel. The peaks can be observed even when the board is directly connected to the PC. However, when using a different operating system, e.g., Ubuntu with a more recent kernel, only one peak emerges in absence of delay.

With delay, the distributions for zeros and ones, as denoted by black and gray, respectively, start drifting away from each other. The distributions separate almost completely when the delay reaches 120 microseconds.

The delay has only a small impact on the performance of our application—the average packet timestamp difference $\overline{\Delta_t}$ is pushed from 128 $\mu s$ to 135.7 $\mu s$. However, this is only because of the way how our computer vision application is implemented. The binarized images are temporarily stored in a first in first out (FIFO) buffer before they are sent to the PC. This explains the effect observable in Fig. 7—as the delay increases, the average time difference of packets without delay becomes smaller than the overall average, and while the average time of packets with delay becomes larger, the overall average time difference $\overline{\Delta_t}$ increases only by a small amount. Multiple peaks arise because the time differences between two packets become dependent on the delay in preceding packets.

Table 1 shows the error rate for all captured datasets. At higher delays, it is possible to




(a) Delay = 0 $\mu s$.

(b) Delay = 40 $\mu s$

(c) Delay = 80 $\mu s$

(d) Delay = 120 $\mu s$

Figure 7: Distributions of time differences for different delays. Zeros are denoted by black frequency bars, ones are denoted by gray frequency bars. In case of an overlap, the bars stack on top of each other. $\overline{\Delta_t}$ denotes the empirical mean of the time differences. Each graph was computed from 20k consecutive bits sent from the FPGA board to the PC over the timing channel.

recover the data sent over the timing channel with a higher confidence. The delay can be adjusted depending on desired performance of the application and the desired confidence of the watermark verifier.

Table 1: Error rate depending on the timing delay

| Timing delay ($\mu s$) | 0 | 20 | 40 | 60 | 80 | 100 | 120 |
|---|---|---|---|---|---|---|---|
| Error rate | 0.5047 | 0.3440 | 0.2682 | 0.2521 | 0.0936 | 0.0953 | 0.0583 |

## 5 Conclusion

In this paper we introduce a new class of IP protection for embedded systems using timing channels. In contrast to previous side channel watermarking schemes (Becker et al., 2011, 2010), timing analysis does not necessarily need laboratory equipment and can be conducted remotely. We propose protocol schemes for both: an authorship and a fingerprint mark. Experimental evidence for this proposal is provided by an implementation on an Altera DE2-70 FPGA board. Using single power and EM traces, conditional timing delays can be reduced to two clock cycles. For network measurements a conditional timing delay of 120 $\mu s$ leads to an error rate of 5.83% while only slightly decreasing the overall performance by 8 $\mu s$. An adaptive adversary aiming at blocking the timing channel for remote detection is forced towards enhanced costs in time, memory and circuitry. We are confident that the proposed methods are indeed applicable in real-world solutions for protecting the IP of hardware and software components and a first step towards remote detection of IP infringement. Future work will study practical implementations of these schemes in embedded software and its degree of robustness on transformation attacks to the embedded watermark.

#### Acknowledgement

This work has been supported in parts by the German Federal Ministry of Education and Research (BMBF) through the project DePlagEmSoft, FKZ 03FH015I3.

## References

Aciicmez, O., Seifert, J.-P., and Koc, C. K. (2006). Predicting Secret Keys via Branch Prediction. Cryptology ePrint Archive, Report 2006/288. http://eprint.iacr.org/.

Aycock, J. (2006). *Computer Viruses and Malware*. Springer.

Becker, G. T., Burleson, W., and Paar, C. (2011). Side-Channel Watermarks for Embedded Software. *9th IEEE NEWCAS Conference*.

Becker, G. T., Kasper, M., Moradi, A., and Paar, C. (2010). Side-channel based Watermarks for Integrated Circuits. In Plusquellic, J. and Mai, K., editors, *HOST*, pages 30–35. IEEE Computer Society.

Bernstein, D. J. (2005). Cache-timing attacks on AES. Technical report.

Boyd, C. (2003). *Protocols for authentication and key establishment*. Springer.

Cannière, C. (2006). Trivium: A stream cipher construction inspired by block cipher design principles. In Katsikas, S., López, J., Backes, M., Gritzalis, S., and Preneel, B., editors, *Information Security*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer Berlin Heidelberg.

Collberg, C. S. and Thomborson, C. D. (1999). Software watermarking: Models and dynamic embeddings. In Appel, A. W. and Aiken, A., editors, *POPL*, pages 311–324. ACM.

Cox, I. J., Miller, M. L., Bloom, J. A., Fridrich, J., and Kalker, T. (2008). *Digital watermarking and steganography*. Elesevier Inc.

Kocher, P. C. (1996). Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Koblitz, N., editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer.

Kocher, P. C., Jaffe, J., and Jun, B. (1999). Differential Power Analysis. In Wiener, M. J., editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer.

Mangard, S., Oswald, E., and Popp, T. (2007). *Power Analysis Attacks*. Springer.

Murdoch, S. J. and Danezis, G. (2005). Low-Cost Traffic Analysis of Tor. In *IEEE Symposium on Security and Privacy*, pages 183–195. IEEE Computer Society.

Nagra, J., Thomborson, C. D., and Collberg, C. S. (2002). A Functional Taxonomy for Software Watermarking. In Oudshoorn, M. J., editor, *ACSC*, volume 4 of *CRPIT*, pages 177–186. Australian Computer Society.

Page, D. (2002). Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel. *IACR Cryptology ePrint Archive*, 2002:169.

Vleck, T. V. (1990). Timing Channels. http://multicians.org/timing-chn.html.

Wang, X., Chen, S., and Jajodia, S. (2005). Tracking Anonymous Peer-to-Peer VoIP Calls on the Internet. In Atluri, V., Meadows, C., and Juels, A., editors, *ACM Conference on Computer and Communications Security*, pages 81–91. ACM.