Privacy-Enhanced Robust Image Hashing with Bloom Filters

Uwe Breidenbach[1,*], Martin Steinebach[2] and Huajian Liu[2]

[1]*TU Darmstadt, Germany*
[2]*Fraunhofer SIT, Germany*
E-mail: uwe.breidenbach@posteo.de; steinebach@sit.fraunhofer.de; huajian.liu@sit.fraunhofer.de
[*]*Corresponding Author*

Received 01 December 2020; Accepted 02 December 2020;
Publication 10 March 2021

Abstract

Robust image hashes are used to detect known illegal images, even after image processing. This is, for example, interesting for a forensic investigation, or for a company to protect their employees and customers by filtering content. The disadvantage of robust hashes is that they leak structural information of the pictures, which can lead to privacy issues. Our scientific contribution is to extend a robust image hash with privacy protection. We thus introduce and discuss such a privacy-preserving concept. The approach uses a probabilistic data structure – known as Bloom filter – to store robust image hashes. Bloom filter store elements by mapping hashes of each element to an internal data structure. We choose a cryptographic hash function to one-way encrypt and store elements. The privacy of the inserted elements is thus protected. We evaluate our implementation, and compare it to its underlying robust image hashing algorithm. Thereby, we show the cost with respect to error rates for introducing a privacy protection into robust hashing. Finally, we discuss our approach's results and usability, and suggest possible future improvements.

Journal of Cyber Security and Mobility, Vol. 10_1, 97–132.
doi: 10.13052/jcsm2245-1439.1014

Keywords: Privacy-preserving, robust hashing, perceptual hashing, hashing, fingerprinting, forensics.

1 Introduction

Today, users store vast amounts of data like music, videos, and images. Advancements in technology allow an easy way to create and store multimedia data. Nowadays, decent cameras are integrated into almost every smartphone. This makes it inexpensive and easy to take plenty of pictures.

Huge collections of images issue a challenge for law enforcement agencies in a forensic investigation to determine if a suspect is in possession of illegal images like child pornography. A manual analysis of every stored image on a suspect's computer system – including external storage – is very time-consuming and impractical. Moreover, it is morally questionable, because of an invasion of privacy. The privacy of the suspect – presumed innocent until proved guilty –, of all other users sharing the investigated computer, as well as of the known victims, who's images form the reference database, is at risk.

The privacy issue is circumvented by using automated tools for the analysis in a forensic investigation. Only pictures marked as suspicious by the tool need further manual investigation. A common approach to detect known illegal images is by using a *cryptographic hash* function. The cryptographic hashes of all the images are matched against a database of hashes of known illegal images, basically doing a blacklist lookup. This approach is efficient and precise in terms of false positives [30]. However, only exact copies are detectable. This is particularly problematic for digital images, where only the visual content, not the data representation matters. Even a marginal alteration during image processing produces an entirely different cryptographic hash value [29, 30]. An image with the same perceptible content is thus not detected correctly.

Robust hashes are introduced to respond to the issue of undetectable images due to slight alterations. *Robust image hashes* compare the perceptible image content, instead of the binary data [29, 30, 32]. An abstract representation of the image content is mapped into the robust hash. Similar images produce similar robust hashes. This is necessary to compare modified images. However, because of this property, drawing conclusions about an image's structure from its robust hash is possible [32, 34]. Potentially private information can thus be revealed, which poses a privacy risk. Figure 1 shows

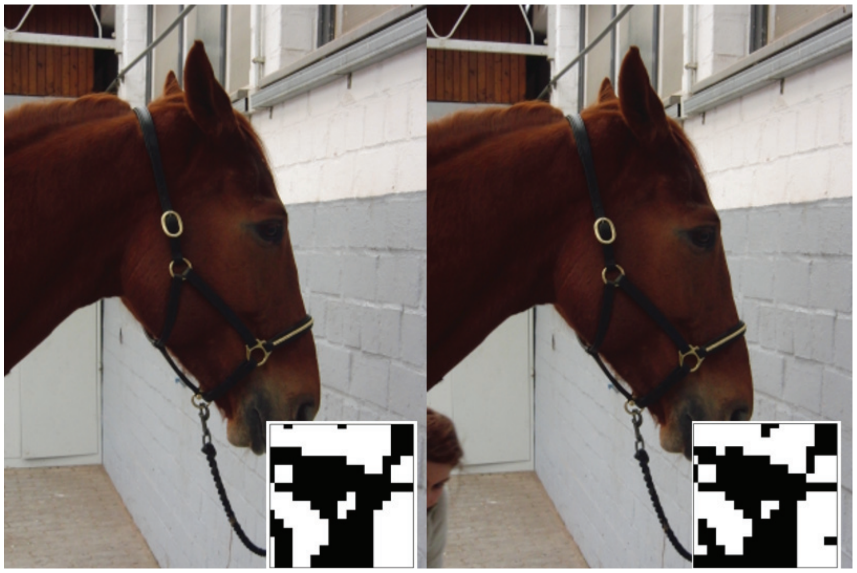

Figure 1 Two similar images feature similar hashes. A robust hash therefore may cause the risk of leakage of image content.

an example. The two images differ at the lower left, still both hashes are similar.

The prosecution of possession and distribution of illegal content is mandatory for law enforcement. Meanwhile, the invasion of privacy needs to be kept at a minimum for everyone involved. An investigator should not be able to gain any information of a suspect, other than knowledge of positively identified illegal images, or the absence of such. Likewise, the private data of the known victims should not be available to anyone.

This demand of privacy is equally true for a preemptive data analysis, e.g., as part of a corporation's content filter to protect their employees and users. The right to privacy is stated in *United Nations*' *Universal Declaration of Human Rights*, and is put into federal law [33]. Therefore, a privacy-preserving technique to efficiently identify known illegal content is imperative. Our goal is to introduce such a privacy-preserving approach.

2 Related Work

This section gives an overview of relevant research regarding our topic of privacy-preserving forensics. Additionally, we briefly introduce the building blocks of our approach, i.e. cryptographic, multimedia, and robust image hashing, as well as the Bloom filter data structure.

2.1 Privacy-Preservation in Forensics

Privacy-preserving forensics was researched in general by [1, 17, 22], among others [1, 17, 22]. In addition, more specific research in computer forensic has been conducted. For example, [2] introduced a privacy-preserving technique of analyzing emails [2]. However, the unique properties of multimedia data has not been considered.

Cryptographic hashes, with their fragility to the smallest alterations, represented an efficient means of bitwise file comparison. However, their fragility is obstructive for processed multimedia files. Multimedia data is processed frequently; while the file is altered, the perceptible content is preserved. This results in false negatives [30]. In contrast, *robust hashes* use the perceptible media content to identify files more accurate [30].

The focus of this research paper is on re-identifying known images in the context of a forensic investigation. Re-identifying images is possible with both, cryptographic and robust hashes. Efficient robust hashes have an advantage for excluding images from further inspection with a whitelist. This is, because known uninteresting images are more likely unaltered, e.g. images shipped with the operating system. Blacklisted – known illegal – images, however, benefit more from robust hashes, since the probability of detecting modified images increases. Note: We re-identify the images itself; we do not identify anything depicted on the images, e.g. a person.

This work focuses on robust image hashes. They show greater potential of identifying known illegal images during a forensic investigation than cryptographic hashes. Although, robust hashes are computationally more expensive, [30] showed they are sufficiently fast for use in forensics [29, 30]. However, as mentioned before, robust hashes suffer from data leakage [32]. From a privacy perspective, this is not acceptable, and needs to be resolved. Even though, we study robust image hashes, our results should be adaptable to different methods and media types.

[32] addressed the privacy issue by combining the privacy-protection of cryptographic hashes, and the robustness against image manipulation of robust image hashes [32]. Firstly, they calculated a robust hash of an image, and identified *weak* bits that are likely to flip due to image alterations. Secondly, multiple versions of the robust hash are created by permuting weak bit combinations. Finally, each permuted robust hash version is hashed

cryptographically. The information leakage is averted, because the cryptographic hashes, stored for future reference, are infeasible to invert. As to privacy-protection, the results are very promising. However, the authors stated that this approach does not achieve a comparable precision, robustness, and efficiency as the robust hash it is based on [32].

In our research we aim to fill the gap among robustness, privacy, and efficiency of current image detection solutions in computer forensics.

2.2 Cryptographic Hashing

Cryptographic hash functions are commonly used in various secure protocols as cryptographic primitives. We only give a brief recap here. For an exhaustive description we refer to the literature, e.g. [12, 25, 27]. A hash function is a deterministic function that maps an input of arbitrary length to a fixed-size hash value. Cryptographic hashes require at least the following properties:

- Efficiency: The hash calculation must be efficient.
- One-way function: Inverting a hash – finding an input to a given hash value – must be computational infeasible.
- Collision resistant: Finding any two inputs hashed to the same hash value must be computational infeasible.

Examples for cryptographic one-way hash functions include the aged MD5 and SHA, as well as modern, unbroken SHA3 and BLAKE2b specification.

2.3 Robust Hashing

Robust or perceptual hash functions use important characteristics of the perceptual content of multimedia data to generate a hash. This hash is robust against certain data manipulations, and can thus identify even transformed media files. Three major properties of robust hashes are defined by [28]:

- *Robustness*: The hash values of perceptually similar multimedia items are to a certain extent invariant. That is, two perceptually similar multimedia files should ideally produce the same – or at least a very similar – hash. The robustness property should hold true for manipulations of the media data with the intend to prevent a re-identification of the file – an attack.
- *Pairwise independence*: Perceptually distinct media items should produce distinct hash values to avoid collisions.

- *Search efficiency*: Querying stored hash values to identify similar hashes to the query needs to be efficient. For some applications, e.g. with large hash database and low latency demands, a linear search ($O(n)$) is not sufficient.

The robustness property is responsible for leaking information. Because similar multimedia data produces similar hashes, it is possible to find an input that creates a similar hash value to a given hash. Therefore, drawing certain conclusions about the media content from its robust hash is possible. For example, portrait photos with a centered face in front of a light plain background, all produce robust hashes with a similar structure.

Steinebach et al. [30] demonstrate this issue with two distinct landscape pictures with similar image structure that result in a similar robust hash [30]. They introduce a mitigation to make the resulting robust hashes more diverse by splitting the image into quadrants, and subsequently generating sub-hashes of each quadrant. Steinebach et al. [32] illustrate the issue with another example in [32]. Thereby, they clarify that it is possible to learn about the structure of an unknown image to a given hash by comparing the hash to hashes of known images. This data leakage poses a privacy risk.

2.4 Robust Image Hashing

A robust image hash is a specialized robust hash. The previously introduced requirements – robustness, pairwise independence, and search efficiency – thus apply. Two additional requirements are necessary for the intended use in forensics [30]. Firstly, a low error rate is crucial to avoid costly manual inspections. Secondly, the hash generation needs to be highly efficient. Ideally, the hashes can be generated on-the-fly, without noticeable latency.

In this paper we want to address the privacy issue of robust hashes, which is particularly critical in the context of a forensic investigation. We thus choose a robust image hashing algorithm as basis that is especially designed for the use in forensics. The block mean value based approach *ForBild*, developed by [30] in [30], is the most viable candidate [30]. Although, ForBild is not the most robust method, it provides high detection rates even for image manipulations like conversions, lossy compression, scaling, and rotation. In favor of efficiency, ForBild compromises for slightly lesser robustness. It uses only low-complexity algorithms and image transformations, and is thus very efficient. This makes ForBild more suitable for the time constraints of forensic investigations than other robust image hashes.

The Block Mean Value Hash (BMVH) was introduced by [36] in [36] [36]. The most efficient variant of their four propositions is used as basis for ForBild. However, robustness against image rotation is thus not supported. The algorithm consists of the following steps:

1. Normalize image to a preset size and transform into grayscale.
2. All pixels of the image I are divided into N non-overlapping blocks I_i, with $0 \leq i < N$, and hash size N (e.g. 256 bit).
3. Calculate the mean value M_i of all pixels for every block I_i, respectively. Subsequently, determine the median value M_d of the mean value sequence M_i.
4. Normalize the mean value sequence into a binary form. The hash value h is calculated by setting each bit h_i either to 0 if $M_i < M_d$, or 1 if $M_i \geq M_d$.

With ForBild, [30] extend the BMVH algorithm with a few simple improvements to circumvent identified issues [30]. For one, the image is split into four quadrants, each with own mean value M_d. For images with similar structure, this results in more diverse hashes. Secondly, a robustness against mirroring is introduced by mirroring the image during hash generation, such that the darkest quadrant is placed in the upper left corner. Thirdly, ForBild introduces weighted bits that indicate how likely a bit flips, due to image transformations. By calculating the similarity with weighted bits, the precision can be improved for inconclusive cases. This last improvement, however, is not compatible with our approach.

One important aspect of the ForBild hash behavior is that the flipping of hash bits is not linear. Figure 2 shows an example. The hashes at weak and medium JPEG compression flip at different positions. The strong JPEG compression then combines positions of both, weak and medium hashes, and adds more flipped positions. This means that flipping positions is chaotic and hard to predict. A maximum assumption can lead to reduction of the ability to distinguish between images. A minimum assumption may not include flipping positions that occur at weak changes, leading to an unexpected increase of false negative results.

2.5 Zero-Knowledge Protocol and Secure Multiparty Computation

A *zero-knowledge protocol* or *zero-knowledge proof* (*ZKP*) is a specific kind of *secure multi-party computation* (*SMC*). In a zero-knowledge protocol one

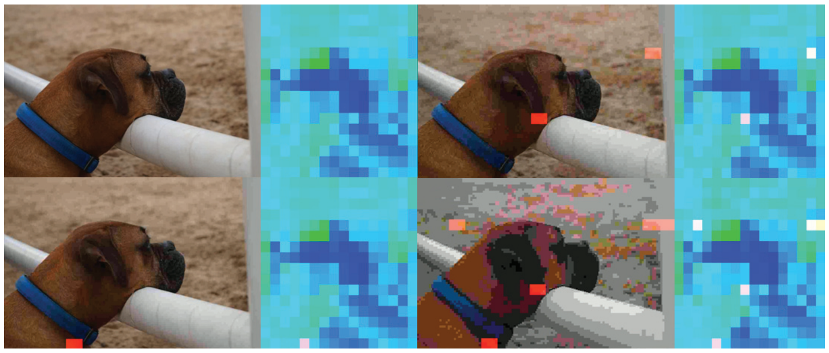

Figure 2 Four variants of one image and the respective robust hash. Up left: original image. Low left: weak JPEG compression. Up right: medium JPEG compression. Low right: strong JPEG compression.

party proofs to another party, that the former has knowledge of a secret value or algorithm. This would be a trivial task, by simply revealing the secret, if it were not for the zero-knowledge part. The challenge lies in proving it without revealing any information other than the fact of knowing the secret itself. An SMC in general is the task of computing a function jointly between multiple parties over their respective input [7,8]. Meanwhile, neither party is revealing its private inputs. This also means that no trusted third party is necessary for the computation. For this paper the secure computation is between two parties only – the application and the database. The focus thus lies on *secure two-party computation* (*S2PC* or *2PC*).

ZKP and SMC can be interactive and non-interactive [11]. For an interactive ZKP or SMC all involved parties are communicating with each other to reach the goal. In a non-interactive protocol there is no further interaction after an initial message. All the necessary information to fulfill the task is thus shared between the parties in one message. Usually, the non-interactivity neither reduces the computational nor communicational complexity of the method. Only the online phase, where all parties interact with each other, is reduced to the initial communication.

The securely computed functions of SMC are manifold. The purpose of ZKP is to proof the knowledge of a value, or a solution to a problem, to another party. Many SMC protocols are specifically designed for one purpose only, e.g. solving the Millionaires' problem[1] [37], or calculating the

[1]The Millionaires' problem is a classic SMP problem: Two millionairs want to figure out, which of them is richer, without revealing their fortune. In general, it is the problem of solving the equation $a \leq b$, without revealing the numbers a and b.

Hamming distance [7, 8, 19]. These SMC protocols are referred to as *special purpose SMC*. In contrast, there are *general purpose SMC* protocols, that are able to calculate multiple functions (of a specific class). The function can be passed as part of the protocol's initialization, or specified in advanced (before starting the actual protocol). Generally, special purpose SMP protocols are more efficient for the same task, than general purpose SMP protocols.

The idea of SMC arose in the 1980s from the work of [37] and [15]. Three major approaches exist today on how SMC is realized. New approaches are often based on one of these three techniques, by extending or combining them.

One, *homomorphic encryption* belongs to the first proposed protocols ever [11, 13, 26]. Homomorphic encryption is an encryption scheme that allows (some) computations on the cipher text (most common: addition). The computational complexity is relatively high, and the functionality of the protocols is limited [8].

Two, *Yao's garbled circuits* introduced protocols with a better performance and wider array of capabilities [37]. The function, which should be computed jointly and securely, is described as a Boolean circuit known to both involved parties. One party garbles (encrypts) the circuit and the other evaluates (decrypts) it, each with their private input. The obtained value is encrypted, and can be decrypted by the first party. For the communication, oblivious transfer is used.

Three, *oblivious transfers* (*OT*) allows a sender to transfer a piece of an information privately to a receiver [24]. The sender remains oblivious about which piece of the information the receiver receives. OT is already used in some SMC protocols like Yao's garbled circuits [37]. Yet, more recent research reveals the existence of significantly more efficient OT implementations [9, 18].

2.6 Bloom Filter

A *Bloom filter* (BF) is a probabilistic data structure invented in [5] by [5] [5]. BFs are space and time efficient, due to the introduction of a small error margin. BFs can efficiently determine an element's membership in a set. Adding elements to a BF, and testing elements memberships is, with a computational complexity of $O(1)$, independent of the number of elements contained in the BF [4].

Internally, a BF uses a zero-initialized bit array $BF[]$ of size m. The universe set of elements is denoted by U, the set of contained elements by

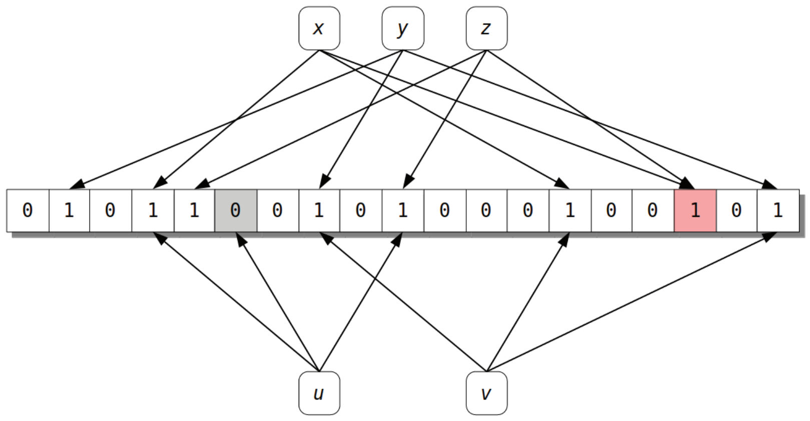

Figure 3 A Bloom filter with $m = 19$, $k = 3$, and $S = \{x, y, z\}$. The arrows point to the bit positions in the $BF[]$ that are set by the elements, respectively. A bit collision of elements x and z is highlighted red. Element $u \in U$ is not a member of BF, which is indicated by the gray highlighted $\bar{0}$bit. Although all bit positions of element v are set, it is actually not a member of BF, and thus $v \in V$ is a false positive.

S, and the element capacity by $n = |S|$. To insert an element, it is hashed with k different hash functions. The resulting hashes are used to index a bit positions in $BF[]$, which subsequently are set to 1. Testing an element's set membership is analog to adding one, except that all k identified bit positions are checked if they are set or unset. Only if all bits are set, the element is – most likely – contained in the BF.

The probabilistic virtue of BFs derives from a limited array size, and the deliberate occurrence of collisions, while adding new elements. A direct consequence of having collisions is the possibility of false positives during a membership test. This is the price to pay for space and time efficiency. The false positive probability is denoted by p, and the hiding set of false positive elements by V. Optimizing a BF for different properties is possible during initialization by choosing appropriate values for m, k, and p in the equations and approximations provided by [4]. Figure 3 shows an example of a BF.

BFs show a great potential, due to their efficiency and basic security, to efficiently enhance the privacy of a robust image hash.

2.7 Metrics

A robust image hash needs to fulfill different requirements to be practical and reliable for its purpose. In the following, we introduce established metrics to measure our requirements.

Common to all hash functions is, that they map an input of arbitrary size to a hash value of relatively small size. A smaller hash size is more likely to produce collisions. A bigger hash size increases the memory consumption of the database, as well as the data transfer size. Additionally, the hash computation time increases, negligible for some algorithms though. For a robust image hash in forensics, a hash size of 256 bit is appropriate. For a cryptographic hash, the hash size depends mainly on the attack model. A hash size of 256 bit is considered secure for most usage scenarios. For applications with high security criteria, or to be prepared for the future, a hash size of 512 bit is recommended.

The robustness of a robust image hash is indicated by its ability to identify modified images. The *true positive rate* tp and the *false negative rate* fn are appropriate measurements to determine the robustness. A true positive test result means that an element stored in the database is correctly detected as such. A false negative result means that an element actually contained in the database is not found. The recall measures the probability that an item contained in the database is correctly identified as such.

The pairwise independence describes the hash's ability to avoid collisions for perceptually distinct images. The resulting hash values should be distinct as well. To quantify this requirement, the true positive rate and the *false positive rate fp* are used. A false positive result means that an element that is not stored in the database is falsely identified as contained. The precision measures the probability that an identified item is truly stored in the database. It thus is a measure of how reliable the algorithm is.

The search efficiency reflects the time that is necessary to decide if a test image is contained in the database or not. With the algorithm's complexity, the efficiency of a query can be estimated. It is common practice for perceptual hashes that the entire database is searched for a query. The complexity of a search is thus $O(n)$. For huge databases even this complexity could become problematic. Nevertheless, a complexity of $O(n)$ serves as reference.

The efficiency of calculating a robust image hash is of importance as well. This is even true today with all the advancements of computer hardware and increased computational power. That is, because at the same time the size of multimedia and thus image collections increased drastically. The computation time is of relevance for initializing the database, and also for querying test images. However, this metric depends highly on the computational power of the computer system in use.

The *Hamming distance* is a metric to quantify an error between two binary data points. It is often used to determine the similarity of perceptual hashes. The Hamming distance is defined as $HD(x, y)$ the number of bits that differ between binary data points x and y [16]. For example, the bytes 11111111_2 and 11110110_2 differ by two bits. Their Hamming distance thus is 2.

The *Bit error rate* (*BER*) is a normalized metric of the distance of two binary values, in our case hashes. The number of mismatched bits i (also known as the Hamming distance) divided by the hash size n yields the BER [36].

3 Concept

We aim to develop a privacy-preserving robust image hashing application. Such an application consists of three major components: A robust image hashing algorithm (see 2.4), a database to store and query for known image hashes, and a privacy-preserving technology that secures the image hashes. We consider three privacy protection options: One, a zero-knowledge protocol privately queries an external database. Two, a homomorphic encryption protects a local database that is queried in the encrypted domain. Three, a Bloom filter (see 2.6) is used as database to store multiple irreversible cryptographic hashes (see 2.2) for each robust image hash. Figure 4 illustrates our concept idea.

The first option privately generates robust hashes, e.g. of a suspects computer, with a client-side application. The reference database with robust hashes of known images, e.g. child pornographic images, is securely stored externally. The client-side queries the database with a robust hash, and gets only a similarity score of the closest match. The database, on the other side, learns nothing about the queried robust image hashes. A zero-knowledge protocol could realize such a private communication. However, the communicational complexity of existing zero-knowledge solutions is not efficient enough. To query one closest-matching similarity score, the zero-knowledge protocol has to run for each robust hash in the database. This is very inefficient with a large test size, and especially for large databases.

The second consideration, protecting a database with homomorphic encryption, and querying it locally in the encrypted domain, suffers from a similar efficiency problem. Determining the highest similarity score of one queried robust hash, makes it necessary to calculate the similarity for

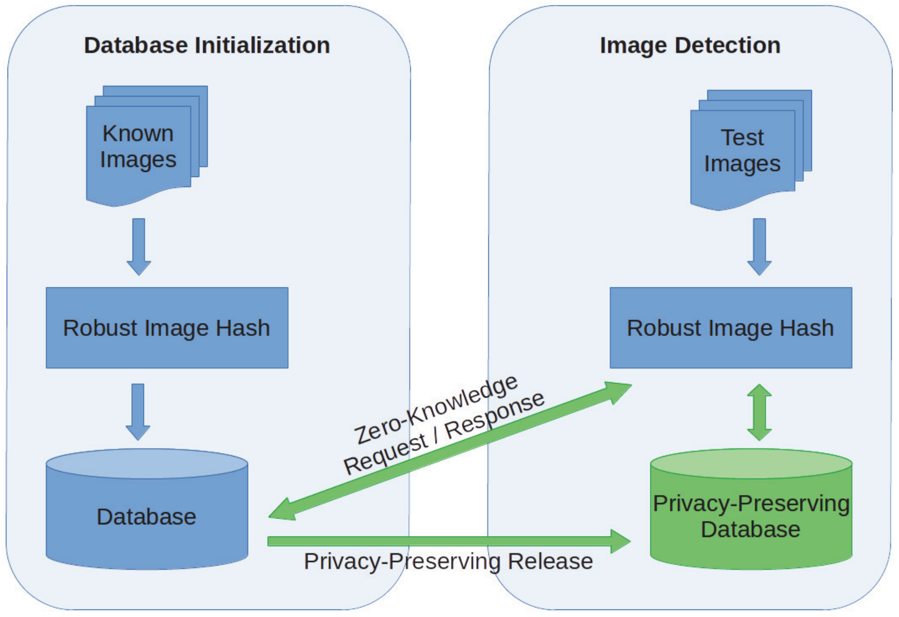

Figure 4 Idea of our image hashing application, split into two modes of operation: initialization (left) and detection (right). A secure and private communication is indicated by the arrows between the modes of operation. Either we have secure and private requests and responses, or we release a privacy-preserved database.

every image hash in the database. The computational complexity thus quickly exceeds a tolerable cost limit in a forensic investigation.

Our third option queries a local, privacy-preserved database as well. We replace the conventional database with a BF. The BF in combination with the one-way property of a cryptographic hash function privately stores the known robust image hashes. The privacy of all elements in the database is thus protected. The challenge is to establish a membership test for similar items. That is for one, because cryptographic hashes match only exact copies, and secondly, because BFs only support exact membership tests.

We tackle this challenge by *partitioning* each robust hash into multiple *blocks*. Subsequently, each block is cryptographically hashed and stored separately in the BF. For similar images, and thus similar robust image hashes, only a few blocks should differ. The similarity of a robust hash can therefore be determined by the number of hits and the total number of tested blocks. To improve robustness (for the cost of space and time efficiency), robust hashes can be differently partitioned and stored multiple times.

3.1 Application Flow

A more detailed overview of the application flow can be formalized by the following. We highlight the involved elements of each step. We start with the process of adding an image to the database:

1. A *robust hash* of an *image* is obtained using ForBild (see 2.4).
2. Split the robust hash into multiple *blocks* by using at least one *partition*.
3. Cryptographically hash (see 2.2) each *block*.
4. Split each *cryptographic hash* into multiple *indices*. This optimization is possible, because $BF[]$ has at maximum a 64 bit index, and modern cryptographic hashes have a typical size of 512 bit.
5. Normalize each *index* to the integer interval of $[0, m - 1]$.
6. Set all the bits in $BF[]$, mapped by the normalized indices.

Testing an image is analogous to adding one, except for:

6. Check all the bits in $BF[]$, mapped by the normalized indices.
7. A *block* is detected only if all its bits, and in extend all indexed bits of a *cryptographic hash*, are set. A *block* is not identified if only one bit is unset.
8. Calculate the *robust hash*'s *similarity score*, using the results of all tested *blocks*.

3.2 Second Decision Stage

We consider a second detection stage to analyze inconclusive results further. The idea is to determine the closest matches in the database to a robust image hash. With this reduced set of closest matches, a different method of identifying robust image hashes with a high complexity becomes viable, e.g. zero-knowledge or homomorphic encryption.

The majority of images should be identified on the client-side using our privacy-preserving BF. In practice most images should be unknown, and thus unambiguously tested negative. In ambiguous cases the client sends a *flat BF*, i.e. all normalized indices of the robust image hash in question, to a server.

The server holds a BF like structure, initialized with the same parameters as the client-side BF (more precisely: it is the other way around; the privacy-preserving client-side BF derives from the server). The difference lies in the internal bit array, which is replaced on the server-side by a list of image IDs for each set bit. By looking up and counting all IDs indexed by the flat BF query, the server can determine the closest matches. Finally, with this drastically reduced subset of the original database of known images, an additional decision algorithm of high complexity, e.g. zero-knowledge based, can be initiated.

4 Implementation

This section briefly describes the implementation of all necessary components. We implement our approach efficiently in modern C++17 standard.

4.1 Cryptographic Hash

Our main concern regarding a cryptographic hash is its security, because we one-way–encrypt the blocks of a robust image hash with it for privacy protection. We thus select a few established candidates that are considered secure. *SHA2* and *SHA3* are United States' Institute of Standards and Technology (NIST) standards [12]. As a SHA3 standardization finalist, *BLAKE2b* provides state-of-the-art security [23].

Efficiency is a subordinated requirement for our choice. We thus test our candidate's efficiency (256 and 512 bit if available), using two widely used, modern C++ libraries, crypto++ and botan$\bar{/}$2.[2] The most efficient[3] – and thus our choice – is the botan$\bar{/}$2 implementation of BLAKE2b with 512 bit.

Our selection, BLAKE2b, has a hash size of 512 bit. Since our index uses at maximum 64 bit, we can improve the robustness of our approach by using $512 \text{ bit}/64 \text{ bit} = 8$ indices to store a block. The computation overhead is negligible. However, the BF memory requirement increases.

4.2 Robust Image Hash

As described in section 2.4, we choose *ForBild* as robust image hashing algorithm. A C++ implementation is provided by the *rHash* API, developed at Fraunhofer Institute for Secure Information Technology (SIT).

4.3 Bloom Filter

A Bloom filter is the key component of our implementation. An overview of the BF functionality is given in section 3.1. This section describes the implementation in more details.

[2]An additional advantage of botan$\bar{/}$2 is its security review and improvement by the German *Federal Office for Information Security*.

[3]Even though popular SHA1 and MD5 are excluded, because they are considered broken, they are outperformed by BLAKE2b anyways.

The BF is initialized with the maximum capacity of stored robust image hashes n. We optimize the internal array size m for space efficiency and privacy by approximating:

$$m' = \frac{n * P * k * k_2}{\ln 2}, \tag{1}$$

with the number of partitions $P \geq 1$, the number of blocks per partition $k = 16$, and the number of indices per block $k_2 = 8$. The best privacy capability is reached with 50 % set and 50 % unset bits of $BF[]$. Finally, we set the size m to a prime equal to or greater than m'. All initialization parameters are fixed and can not be changed later, without regenerating the whole database.

After initialization, the BF can process image hashes. An image hash is partitioned into multiple blocks with at least one partition. Because partitions highly affect privacy, robustness, precision, and efficiency, we choose the partition parameters very carefully. A block size of 16 should provide enough data variety to prevent brute forcing the pre-image of a cryptographic hash, while being small enough to provide sufficient robustness against image modifications.

A block thus consists of a 3 bit partition ID, a 4 bit block position, and 16 bit (block size) of robust image hash data. That sums up to a total of 23 bit per block. During our evaluation, we vary the partition combinations to optimize robustness and precision. Examples of partitions are shown in Figure 5.

Subsequently, each (23 bit) block is cryptographically hashed, what eventually results in the indices. Each index needs to be normalized to a value that fits into $BF[]$, i.e. $[0, m-1]$. To ensure a uniform distribution of the indices, we use the *division method* given by equation:

$$h(x) = x \bmod m \tag{2}$$

We circumvent an error-prone case – a reduction of $h(x)$ to the s least significant bits, e.g. by setting $m = 2^s$ – during initialization by choosing m prime.

The similarity score s_{sim} of an image hash to the closest one stored in the database is statistically determined with:

$$s_{sim} = 1 - \frac{blk_{neg} * 0.5}{blk_t}, \tag{3}$$

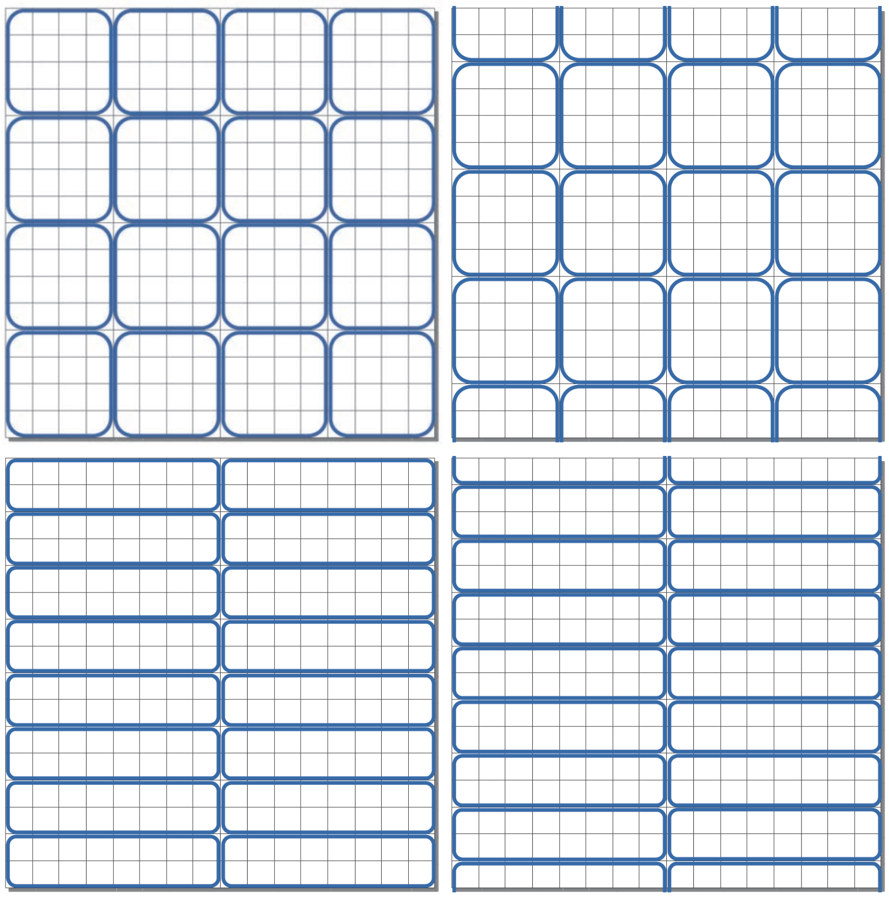

Figure 5 A partition separates a robust image hash into multiple blocks. The two partitions on top are more fine-grained regarding perceptible blocks, while the lower two are more efficient, using consecutive data bytes.

with $0.5 \leq s_{sim} \leq 1$, the total number of tested blocks $blk_t > 0$, and the number of negatively tested blocks $0 \leq blk_{neg} \leq blk_t$. A $s_{sim} = 0.5$ constitutes the average similarity compared to a randomly chosen bit sequence.

4.4 Bloom Filter Hardening

The security and privacy properties of BFs are studied in e.g. [3, 4, 20]. Additionally, several hardening techniques for BFs are sugested in [10, 21], such as salting, noise injection, and a random selection of hash functions. Some of which are useful to improve our approach's security.

A collusion attack analyzes different releases of a BF database in order to identify newly added image hashes. The stored hashes are still one-way encrypted. Thus, to determine if a block belongs to an image hash, the block has to be probed, i.e. cryptographically hashed and used as normalized indices to verify set bits of $BF[]$.

To mitigate collusion attacks, we make the comparison harder. Firstly, never reusing the same BF size prevents a direct bitwise comparison. A blacklist of already used BF sizes is thus kept. Secondly, a BF is initialized with a random unique salt. Half of the salt is prepended and half is appended to each block before cryptographically hashing it. A salt makes probing blocks in a large scale infeasible, because it prevents precomputed rainbow tables. The disadvantage of using a salt and a unique BF size is that the BF needs to be regenerated on every update, not only when the capacity is insufficient.

However, in case of a successful attack, an identified image can be denied due to the probabilistic nature of BFs. Because image hashes share bits in $BF[]$ due to collisions, the possibility of false positives exists. By increasing the false positive probability, elements can be denied with higher plausibility. In return, the BF size needs to be adjusted to compensate for a decrease in precision and robustness.

One technique to increase the deniability is to inject random noise. A second technique is the injection of tailored false positive image hashes that share a lot of bits with true positive elements. We implement tailored false positives by splitting two known images in half and combining them to two fake images. Since a lot of consecutive image data is unchanged, so are many stored blocks. The additionally added data is thus limited. However, true positive blocks are more plausible deniable, because they are covered by false positive elements. This method proved to be more effective than adding random noise, and is thus pursued [4].

4.5 Closest Match Bloom Filter Extension

We modify our BF implementation to determine the closest matches of a requested robust image hash on the server-side (see 3.2). The server's internal BF array is replaced by a list of the same size. This list holds a linked list for each set bit. Each linked list stores all IDs of corresponding known images.

A request is made by the client with a flat BF, a space efficient BF with only one element, i.e. a list of normalized BF indices of a robust image hash. The server determines all indexed linked lists, and thus all indexed IDs of known images. The quantity of occurrences of each ID indicates the similarity. The higher the frequency of an ID, the more indexed blocks are identified, and thus the closer the match.

Our closest matching BF extension provides a proof of concept. The implementation makes no claim to be complete or efficient. It merely proofs that an additional higher complexity decision stage can be implemented in the future.

5 Evaluation

The evaluation conditions are initially established in this section. The evaluation is conducted with a set of known and unknown images. Different combinations of partitions with different thresholds are evaluated first to determine sane parameters for our approach. We use these parameters for the remaining evaluation. The most important quantifiers are *precision* and *recall*. The precision is the probability that an identified image actually is contained in the database. It thus measures the reliability. The recall is the probability of positively re-identifying a known image that is stored in the database. It thus measures the robustness.

5.1 Test Set

We aim to evaluate our application in the context of a forensic investigation, e.g. to prosecute child pornography charges, as realistically as possible. Therefore, we settle for the *Galaxy set* of cheerleader photos. This set has been used for its characteristics in related research, e.g. [6, 29–32, 35]. All photos show one or more people with diverse skin colors. The pictures were taken inside and outdoors, with different resolution, and in landscape and portrait orientation.

We divide 3092 distinct photos randomly into two groups, *known* and *unknown*, of 1546 pictures each. The known group is inserted into the database to evaluate how well images can be re-identified. The unknown group is used to evaluate the false positive rate. It thus is not added to the database.

5.2 Attacks

To evaluate the algorithm's robustness and reliability, we use different image processing techniques to attack our test set. The techniques used, are covered by ForBild. They range from normal image processing, e.g. to reduce the size, to image manipulations with a malicious intent, e.g. to circumvent copyright filters. All 3092 pictures are attacked with the following 15 transformations:

- JPEG compression with quality $q \in \{90, 80, 70, \ldots, 10\}$.
- Scaling 150 %, 110 %, 90 %, 75 %, and 50 %.
- Horizontal mirroring.

ImageMagick version 7.0.9-10 Q16 x86_64 is used for the attacks. JPEG compression is an attack by itself, we thus use lossless PNG for the other attacks. In summery, our evaluation thus consists of:

- 1546 known images in the database.
- 1546 unknown images.
- $1546 * 15 = 23190$ attacked known images.
- $1546 * 15 = 23190$ attacked unknown images.
- 49472 test images in total.

5.3 Testbed

We evaluate our implementation single threaded on a test system with the following specifications:

- Notebook: Lenovo ThinkPad T450s
- Processor: Intel Core i5-5300U, 2.30 GHz
- Memory: 8 GiB DDR3L SDRAM, 1600 MHz
- Storage: 256 GB SSD, Serial ATA-600
- Operating System: Manjaro Linux, kernel 5.4

5.4 Parameter Determination

In the following we determine the most viable parameters for our application, for a later evaluation of its robustness and reliability. Several partitions are provided to separate a robust image hash into multiple blocks (see 4.3). The partitions are designed to complement each other, i.e. any two partitions never define the same block.

A list of partition(s) that work nicely together, is one essential parameter to determine.

The similarity threshold t_{sim}, whether a robust image hash is considered similar or distinct to a known one, is the second crucial parameter to determine for practical use in forensics.

We conduct our evaluation by calculating recall and precision for different t_{sim}, and multiple combinations of partitions. The threshold t_{sim} needs to be considerably larger than 0.5, the average similarity to randomly chosen data. We thus evaluate $0.7 \leq t_{sim} \leq 0.975$ in steps of size 0.025. The following diverse combinations of partitions are used in a BF for evaluation: one, partition 1 only (P1); two, partition 1 and 2 (P1,2); three, partition 3 and 4 (P3,4); and four, partition 1 to 4 (P1,2,3,4).

The results of the evaluation are shown in Table 1. Additionally, graphs 6 and 7 illustrate precision and recall, respectively. A precision approaching

Table 1 Recall and precision of Bloom filters with different combination of partitions and threshold $0.85 \leq t_{sim} \leq 0.975$ in 0.025 steps. The most promising results for our use case in forensics are marked.

t_{sim}	Recall	Precision	Recall	Precision
	P1,2,3,4		P1,2	
0.975	0.566866	0.999145	0.530320	0.999010
0.95	0.765362	0.999050	0.791801	0.998114
0.925	**0.879892**	**0.998257**	0.869542	0.995925
0.9	**0.938753**	**0.993072**	0.943807	0.984565
0.875	0.973763	0.975459	0.974652	0.960518
0.85	0.984031	0.950041	0.981808	0.942085
0.825	0.990742	0.915807	0.991389	0.898772
0.8	0.993895	0.878475	0.993451	0.873121
0.775	0.995310	0.829850	0.995391	0.802490
	P3,4		P1	
0.975	0.517101	0.999063	0.494947	0.998043
0.95	0.779148	0.998860	0.756185	0.995529
0.925	**0.864570**	**0.998086**	0.895294	0.986898
0.9	0.942149	0.989219	0.953105	0.965438
0.875	0.974854	0.970382	0.977927	0.930850
0.85	0.981970	0.956036	0.977927	0.930850
0.825	0.990055	0.916816	0.989044	0.887667
0.8	0.992885	0.893513	0.993370	0.833854
0.775	0.995149	0.822425	0.995674	0.762697

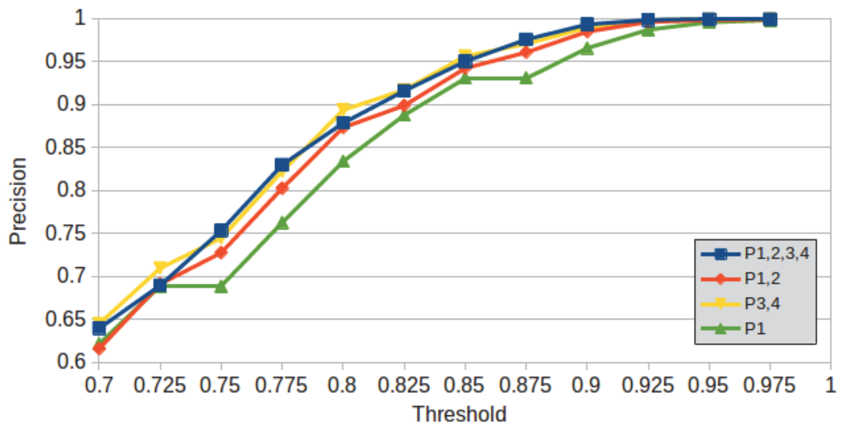

Figure 6 Precision of BF for different partitions and t_{sim}.

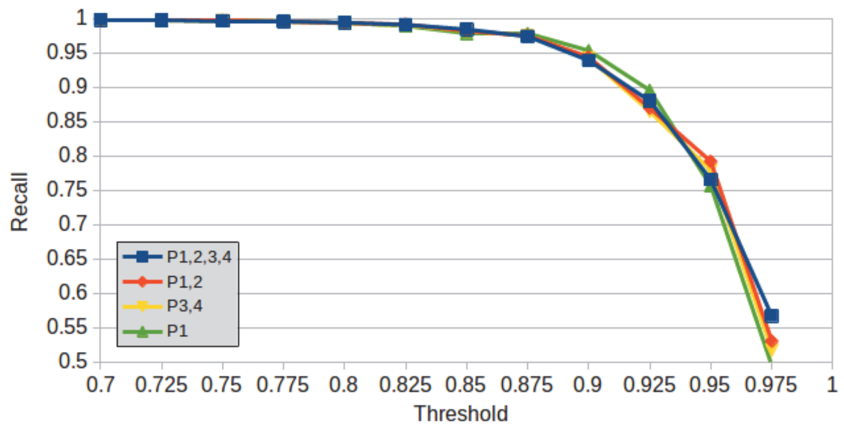

Figure 7 Recall of BF for different partitions and t_{sim}.

1, and a recall greater than 0.99, would be ideal in the context of a forensic investigation [30]. The results show that BFs improve using more partitions. However, the precision and recall of P3,4 comes close to the results of P1,2,3,4. Figure 8 shows a comparison of recall and precision for different thresholds with P1,2,3,4.

The most viable parameter combinations are:

1. P1,2,3,4 with $t_{sim} = 0.925$
2. P1,2,3,4 with $t_{sim} = 0.9$
3. P3,4 with $t_{sim} = 0.925$

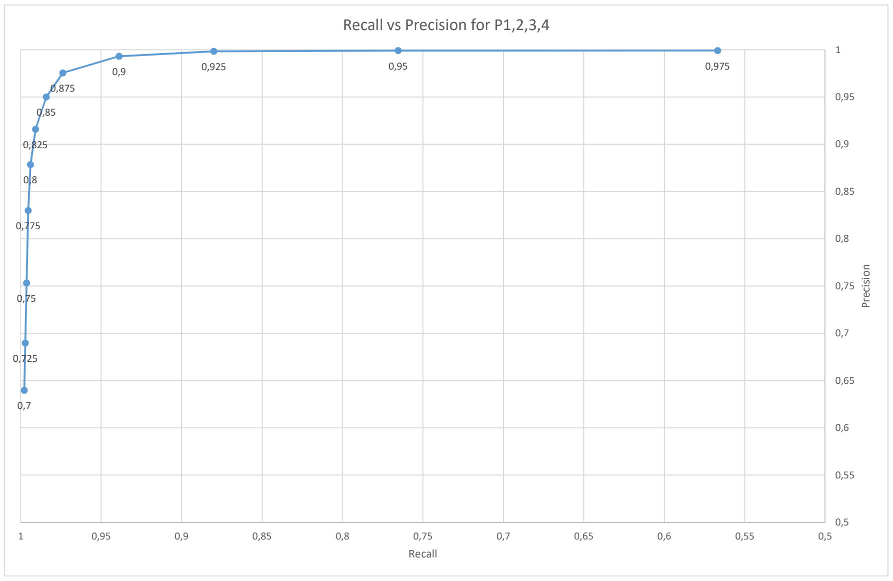

Figure 8 Recall vs. precision for P1,2,3,4. Thresholds are attached to the data points.

Table 2 Shows the conducted efficiency measurements for ForBild and the BF variants, i.e. the database size (*db size*), the database initialization time (*db init*), and the time to query the entire test set (*test*)

Algorithm	db size (KiB)	db init (s)	Test (s)
ForBild	48.4	18.89	691.09
BF P1,2,3,4	139.4	18.57	661.01
BF P1,2	69.7	21.21	709.16
BF P3,4	69.7	20.83	678.98
BF P1	34.8	18.33	667.91

The efficiency measurements are shown in Table 2. Time is measured in high precision with *std::chrono::high_resolution_clock::now()* C++ standard function. The database size is moderately larger for BFs compared to ForBild. The size of a BF rises linear with the number of used partitions. Initializing the BF databases takes slightly longer. However, this should be negligible in practice, since the robust hash generation is still the most time-consuming part.

The elapsed time to test the whole image test set of 49472 images against the database, is usually faster for the BF implementation by a few seconds

Table 3 Performance at different thresholds

Total img	Threshold	TP	FN	TN	FP	Recall	Precision
24736	**0.975**	14022	10714	24724	12	**0.56686611**	**0.99914493**
24736	**0.95**	18932	5804	24718	18	**0.76536223**	**0.99905013**
24736	**0.925**	21765	2971	24698	38	**0.87989166**	**0.99825712**
24736	**0.9**	23221	1515	24574	162	**0.93875323**	**0.99307189**
24736	**0.875**	24087	649	24130	606	**0.97376294**	**0.97545863**
24736	**0.85**	24341	395	23456	1280	**0.98403137**	**0.95004098**
24736	**0.825**	24507	229	22483	2253	**0.99074224**	**0.91580717**
24736	**0.8**	24585	151	21335	3401	**0.99389554**	**0.87847495**
24736	**0.775**	24620	116	19688	5048	**0.99531048**	**0.82985034**
24736	**0.75**	24645	91	16664	8072	**0.99632115**	**0.75327811**
24736	**0.725**	24663	73	13637	11099	**0.99704884**	**0.68964264**
24736	**0.7**	24679	57	10838	13898	**0.99769567**	**0.63973352**

compared to ForBild. This effect should become more noticeable with growing database size n, since ForBild compares elements with complexity $O(n)$, while BF only requires $O(1)$. The difference in query time between BFs do not correlate to the number or type of used partition. It seems other factors have more impact on the overall time consumption, e.g. storage lookup or robust image hash generation.

For our algorithm with use in forensics, precision is more important than recall, and recall is more important than efficiency. We thus select option (1), combining partitions 1 to 4 with $t_{sim} = 0.925$. Table 3 shows the detailed performance of this combination for thresholds from $t_{sim} = 0.7$ to $t_{sim} = 0.975$. The third option – P3,4 with $t_{sim} = 0.925$ – is a good alternative in case efficiency is of greater value.

5.5 Robustness and Reliability

The robustness of our approach – more precisely of the BF using partitions 1 to 4, and $t_{sim} = 0.925$ – against various image manipulations is evaluated by attacking the set of known images (see 5.2). We have 15 attacks – $9\times$ JPEG, $5\times$ scaling, and $1\times$ mirroring – on 1546 known images, resulting in a total of 23190 attacked known images. Subsequently, we test the attacked known images against the database of known images. The resulting recall indicates the robustness.

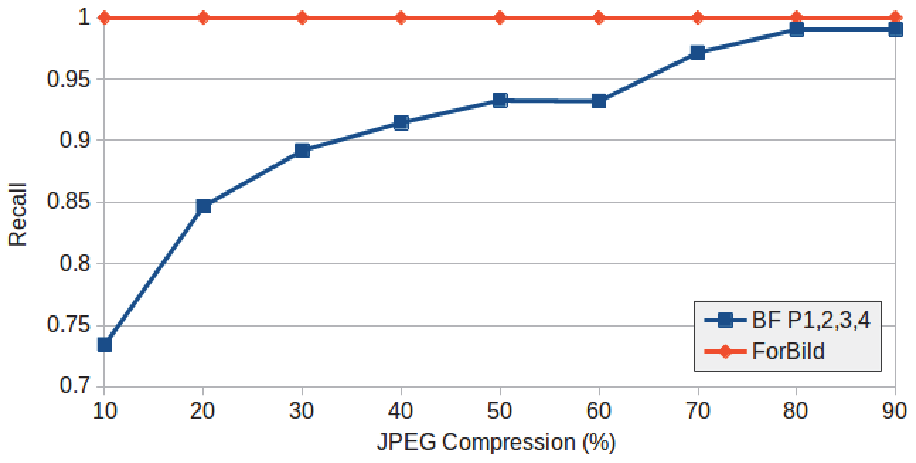

Figure 9 Recall of JPEG compression for different qualities.

5.5.1 JPEG

We evaluate the robustness against JPEG compression for different image qualities. The evaluated image quality ranges in steps of 10 % from 10 % to 90 %. Figure 9 shows the recall for each compression level. The recall decreases with lower image quality and stronger compression rates. The recall falls below 0.9 for qualities of 30 % and below. The average JPEG compression recall is 0.911600. ForBild, in comparison, has a recall of 1.

5.5.2 Scaling

The robustness against image scaling is evaluated by scaling each known image to 50 %, 75 %, 90 %, 110 %, and 150 %. The recall is shown in Figure 10. Scaling severely affects the recall. It is lowered to 0.9 and below. Scaling down is more fragile than scaling up, with a recall below 0.7 for halving the image size. ForBild as reference has a recall of 1 for scaling.

5.5.3 Horizontal mirroring

The robustness against horizontal mirroring is first introduced in ForBild with a recall of 1. The recall of our approach is with 0.76 low.

5.5.4 Robustness Summary

Table 4 lists the recall and precision of all evaluated attacks in greater detail. The issue of decreasing robustness is illustrated by Figure 11. Each dot is

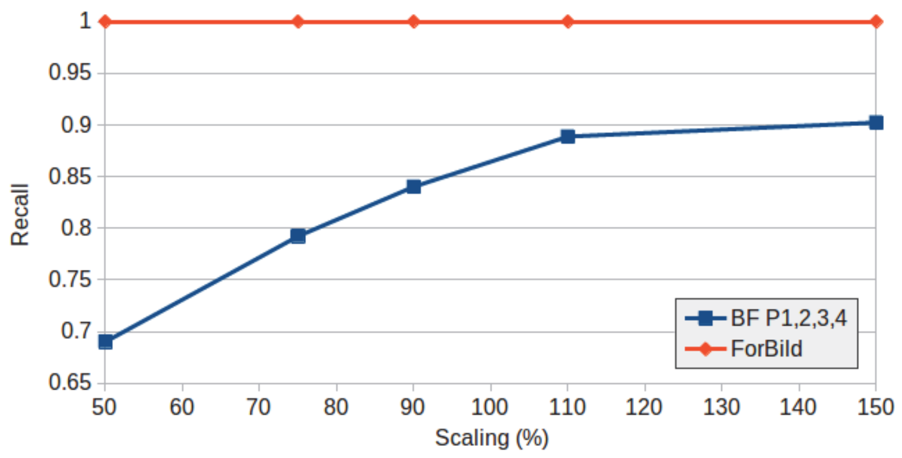

Figure 10 Recall of scaling for different sizes.

Table 4 Recall and precision for all attacks (see 5.2)

Attack	Percent	Recall	Precision
JPEG compression	90	0.990298	0.998695
JPEG compression	80	0.990298	0.998044
JPEG compression	70	0.971539	0.998007
JPEG compression	60	0.932083	0.996542
JPEG compression	50	0.932730	0.997924
JPEG compression	40	0.914618	0.997179
JPEG compression	30	0.891979	0.998552
JPEG compression	20	0.846701	0.997713
JPEG compression	10	0.734153	0.995614
JPEG compression	avg.	0.911600	0.997586
Scaling	150	0.902329	1.000000
Scaling	110	0.888745	1.000000
Scaling	90	0.840233	0.999231
Scaling	75	0.792367	0.996745
Scaling	50	0.690168	0.999064
Scaling	avg.	0.822768	0.999008
Horizontal Mirroring		0.760026	1.000000

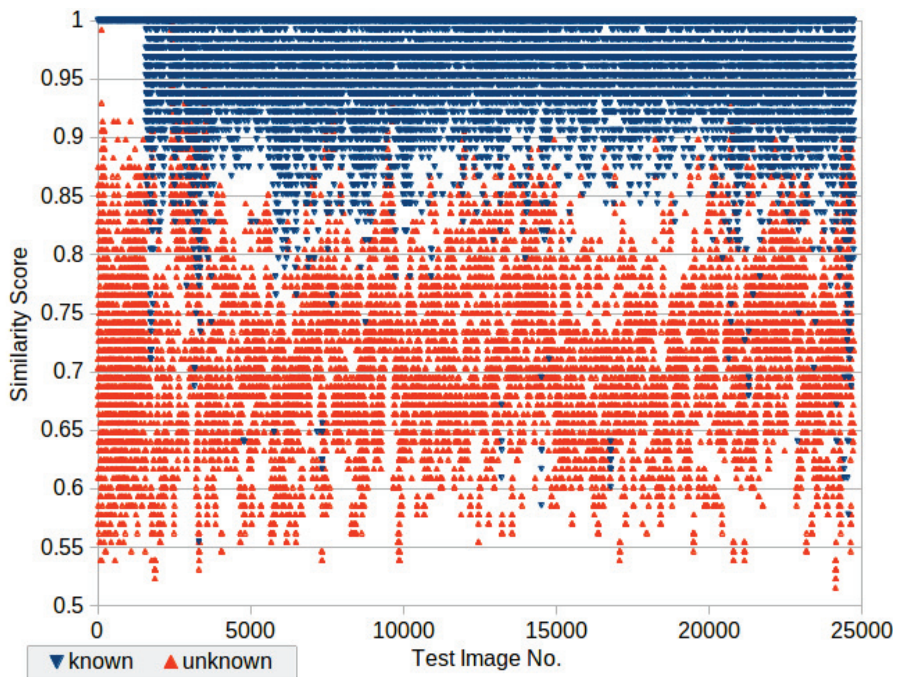

Figure 11 Distribution of the similarity score of known and unknown images.

a similarity score s_{sim} of a picture, all (attacked) known images are blue, while all (attacked) unknown images are red. The lack of a gap between the blue and the red cloud makes deciding between known and unknown more difficult, and thus reduces robustness.

5.5.5 Reliability

We evaluate the reliability of our approach by means of the precision. The precision is the probability that false positives occur. In a forensic investigation, false positives require an expensive manual investigation, and thus need to be avoided. We test the set of 1546 unknown images against the database of known images. This results in a precision of 0.998708 (two false positives exist). Testing the whole test set of 49472 images, including the attacked ones, results in a precision of 0.998257 (38 false positives exist). ForBild, however, provides a precision of 1.

5.6 Privacy

Our approach protects the robust image hashes with a one-way encryption using cryptographic hashes. Additionally, the nature of a BF obfuscates the origin of stored blocks. However, a few attack vectors need to be considered. The most successful attacks on BFs are based on cryptoanalysis, and rely on publicly available identifiers, or a small guessable universe set of identifiers [10, 14, 21]. This does not apply for our approach, since the robust image hashes, and in extend the 23 bit blocks, are not easily guessable. In the following we evaluate the BF hardening techniques presented in 4.4.

The introduction of a unique salt into a BF results in a totally different binary file compared to the same BF without a salt. The changes in robustness and reliability – due to different collisions – are negligible. Initializing and querying the salted BF takes slightly longer. However, this should be irrelevant in practice.

We evaluate two magnitudes of injecting tailored false positives to increased element deniability. [4] introduced a privacy metric for BFs, the γ-K-anonymity [4]. The γ-K-anonymity states that at least $K - 1$ false positives cover any randomly chosen element with probability γ. One round of adding tailored elements gives 0.75-2-anonymity. Two rounds guarantee 1.50-2-anonymity or 0.50-3-anonymity. Overall, the injection of tailored false positives shows only negligible variations in robustness and reliability. This is due to the increase in BF size of approx. 13 and 26 KiB, respectively, to compensate for the elevated false positive probability.

5.7 Closest Match

The closest match server-side BF extension successfully determines the closest matching IDs of known images to a queried robust image hash. The robustness and reliability of the extension is marginally better than of the client-side BF (due to the elimination of false positives). The server only learns the one-way encrypted and normalized indices in the flat BF of the queried robust image hash. The client learns nothing about the known images. However, learning the closest matching similarity score would be allowed.

This proofs the concept of reducing the server-side database size to a small subset of closest matches. By choosing a constant subset size, an algorithm with a complexity dependent on the database size, can be reduced to $O(1)$. A future privacy-preserving implementation of an extra decision stage, e.g. using zero-knowledge, is thus possible. However, the closest match extension needs improvements regarding its efficiency. A database size of almost 8 MiB does not suffice for practical use.

6 Summary and Future Work

The introduced approach is a new attempt to apply privacy protection to robust image hashing in forensics. Our main concern of privacy protection is successfully realized. The robust image hashes, including possible private information, are protected against unauthorized access on both ends, the database and the system under investigation. The confidentiality of the robust hashes is ensured due to the one-way property of cryptographic hashes. It is not computationally feasible to recover any data from the one-way encrypted Bloom filter database.

Additionally, measures to improve on privacy, e.g. salting and γ-K-anonymity, are available at an acceptable cost of efficiency. Further, the proof of concept of a closest match extension lays the ground for an efficient privacy-preserving protocol, e.g. zero-knowledge, to improve robustness and reliability in the future.

The efficiency of a robust image hashing application is important in a forensic investigation. The evaluation results show that our approach is sufficiently efficient, even outperforming the robust image hash it is based on during detection. Especially for larger databases we improve the efficiency, because a hash lookup now is independent of the database size. The provided precision is acceptable for most use cases in forensics.

The robustness against JPEG compression averages at a recall of 0.912. The robustness against scaling and mirroring is lower, with 0.823 and 0.760, respectively. The total average recall lies at 0.880 with a precision of 0.998. The approach does not achieve similar results to ForBild. However, it provides protection of privacy at a cost of 12 % loss of recall.

6.1 Future Work

This work is only one step on the road of making robust hashes privacy-preserving. Our evaluation showed that robustness and reliability of a BF increased by using more partitions. Introducing more distinct partitions increases the number of diverse blocks stored in the BF. This could improve the probability that more blocks survive small bit errors with large impact. Additionally, reducing the block size should show improvements as well. However, this will reduce the privacy guarantee with it.

A promising improvement of our approach is to make the closest match extension more efficient. This could be done by reducing the database size and complexity. The currently used linked lists could be replaced by Bloom filters storing IDs. To determine the corresponding IDs to an index, a BF need to be queried with every known ID. Subsequently, after improving the closest matching implementation, a privacy-preserving communication protocol can be implemented to calculate the similarity, e.g. using zero-knowledge or secure multi-party computation. The protocol's complexity is reduced to a constant one, and is thus efficient. This should improve the robustness and reliability to those of the underlying robust hashing algorithm.

There is a multitude of robust hash function available. Robust image hashing with ForBild was just one example. The presented privacy-preserving solution is independent of the hashed media data. This technique should thus be adaptable to other robust hashes, and possibly other fields of application.

Acknowledgment

This publication derived from the master thesis of the main author [34], supervised by the additional authors. This work is partially taken from it, e.g. 2.5 and 2.7. The master thesis was written at the *Department of Computer Science* of *Technische Universität Darmstadt*, Germany, and the *Media Security and IT-Forensics Group* of the *Fraunhofer Institute for Secure Information Technology*, Darmstadt, Germany, and in cooperation with the *Department of Computer Engineering* of the *Faculty of Engineering* of *Mahidol University*, Thailand. Many thanks for all the help and support!

This research work has been funded by the German *Federal Ministry of Education and Research* and the *Hessen State Ministry of Higher Education, Research and the Arts* within their joint support of the *National Research Center for Applied Cybersecurity*.

References

[1] A. Aminnezhad and A. Dehghantanha. A survey on privacy issues in digital forensics. *International Journal of Cyber-Security and Digital Forensics (IJCSDF)*, 3:183–199, 2014. ISSN 2305-0012. URL https://usir.salford.ac.uk/id/eprint/34016/1/asurvey.pdf.

[2] F. Armknecht and A. Dewald. Privacy-preserving email forensics. *Digital Investigation*, 14:S127–S136, Aug. 2015. ISSN 1742-2876. .

[3] D. Arp, E. Quiring, T. Krueger, S. Dragiev, and K. Rieck. Privacy-Enhanced Fraud Detection with Bloom Filters. In *Security and Privacy in Communication Networks*, volume 254, pages 396–415. Springer International Publishing, Cham, 2018. ISBN 978-3-030-01700-2 978-3-030-01701-9. .

[4] G. Bianchi, L. Bracciale, and P. Loreti. "Better Than Nothing" Privacy with Bloom Filters: To What Extent? In *Privacy in Statistical Databases*, volume 7556, pages 348–363. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-33627-0. .

[5] B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM*, 13(7):422–426, July 1970. ISSN 0001-0782. .

[6] F. Breitinger, H. Liu, C. Winter, H. Baier, A. Rybalchenko, and M. Steinebach. Towards a Process Model for Hash Functions in Digital Forensics. In *Digital Forensics and Cyber Crime*, volume 132, pages 170–186. Springer International Publishing, Cham, 2014. ISBN 978-3-319-14288-3 978-3-319-14289-0. .

[7] J. Bringer, H. Chabanne, and A. Patey. SHADE: Secure HAmming DistancE Computation from Oblivious Transfer. In *Financial Cryptography and Data Security*, volume 7862, pages 164–176. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-41319-3 978-3-642-41320-9. .

[8] J. Bringer, H. Chabanne, M. Favre, A. Patey, T. Schneider, and M. Zohner. GSHADE: Faster privacy-preserving distance computation and biometric identification. In *Proceedings of the 2nd ACM Workshop on Information Hiding and Multimedia Security - IH&MMSec '14*, pages 187–198, Salzburg, Austria, 2014. ACM Press. ISBN 978-1-4503-2647-6. .

[9] S. G. Choi, K.-W. Hwang, J. Katz, T. Malkin, and D. Rubenstein. Secure Multi-Party Computation of Boolean Circuits with Applications to Privacy in On-Line Marketplaces. In *Topics in Cryptology – CT-RSA 2012*, Lecture Notes in Computer Science, pages 416–432, Berlin, Heidelberg, 2012. Springer. ISBN 978-3-642-27954-6. .

[10] P. Christen, R. Schnell, D. Vatsalan, and T. Ranbaduge. Efficient Cryptanalysis of Bloom Filters for Privacy-Preserving Record Linkage. In *Advances in Knowledge Discovery and Data Mining*, volume 10234, pages 628–640. Springer International Publishing, Cham, 2017. ISBN 978-3-319-57453-0 978-3-319-57454-7. .

[11] R. Cramer and I. Damgå rd. Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free? In *Advances in Cryptology*

— *CRYPTO '98*, Lecture Notes in Computer Science, pages 424–441. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-68462-6.

[12] M. J. Dworkin. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Techreport 202, National Institute of Standards and Technology, Aug. 2015.

[13] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-Preserving Face Recognition. In *Privacy Enhancing Technologies*, Lecture Notes in Computer Science, pages 235–253, Berlin, Heidelberg, 2009. Springer. ISBN 978-3-642-03168-7. .

[14] A. Gervais, S. Capkun, G. O. Karame, and D. Gruber. On the privacy provisions of Bloom filters in lightweight bitcoin clients. In *Proceedings of the 30th Annual Computer Security Applications Conference on - ACSAC '14*, pages 326–335, New Orleans, Louisiana, 2014. ACM Press. ISBN 978-1-4503-3005-3. .

[15] O. Goldreich, S. Micali, and A. Wigderson. How to Play ANY Mental Game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM. ISBN 978-0-89791-221-1. .

[16] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, Apr. 1950. ISSN 0005-8580. .

[17] S. Hou, T. Uehara, S. M. Yiu, L. C. K. Hui, and K. P. Chow. Privacy Preserving Confidential Forensic Investigation for Shared or Remote Servers. In *2011 Seventh International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 378–383, Oct. 2011. .

[18] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster Secure Two-party Computation Using Garbled Circuits. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 35–35, Berkeley, CA, USA, 2011. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2028067.2028102.

[19] A. Jarrous and B. Pinkas. Secure Hamming Distance Based Computation and Its Applications. In *RoboCup 2001: Robot Soccer World Cup V*, volume 2377, pages 107–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-540-43912-7 978-3-540-45603-2. .

[20] K. Kanemura, K. Toyoda, and T. Ohtsuki. Design of privacy-preserving mobile Bitcoin client based on γ-deniability enabled bloom filter. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor,*

and Mobile Radio Communications (PIMRC), pages 1–6, Montreal, QC, Oct. 2017. IEEE. ISBN 978-1-5386-3529-2 978-1-5386-3531-5. .

[21] M. Kroll and S. Steinmetzer. Automated Cryptanalysis of Bloom Filter Encryptions of Health Records. In *Proceedings of the International Conference on Health Informatics*, volume 1, pages 5–13, Lisbon, Jan. 2015. SciTePress. ISBN 978-989-758-068-0. .

[22] F. Y. W. Law, P. P. F. Chan, S. M. Yiu, K. P. Chow, M. Y. K. Kwan, H. K. S. Tse, and P. K. Y. Lai. Protecting Digital Data Privacy in Computer Forensic Examination. In *2011 Sixth IEEE International Workshop on Systematic Approaches to Digital Forensic Engineering*, pages 1–6, May 2011. .

[23] M-J. Saarinen and J-P. Aumasson. The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC). Technical Report RFC7693, RFC Editor, Nov. 2015.

[24] M. Naor, B. Pinkas, and B. Pinkas. Efficient Oblivious Transfer Protocols. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 448–457, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics. ISBN 978-0-89871-490-6. URL http://dl.acm.org/citation.cfm?id=365411.365502.

[25] J. R. Nechvatal, E. B. Barker, L. E. Bassham, W. E. Burr, M. J. Dworkin, J. Foti, and E. Roback. Report on the Development of the Advanced Encryption Standard (AES). *Journal of Research (NIST JRES)* -, 106 No. 3, June 2001. URL https://www.nist.gov/publications/report-development-advanced-encryption-standard-aes.

[26] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich. SCiFI - A System for Secure Face Identification. In *2010 IEEE Symposium on Security and Privacy*, pages 239–254, Oakland, CA, USA, 2010. IEEE. ISBN 978-1-4244-6894-2. .

[27] B. Preneel. Cryptographic hash functions. *European Transactions on Telecommunications*, 5(4):431–448, 1994. ISSN 1541-8251. .

[28] J. S. Seo, J. Haitsma, T. Kalker, and C. D. Yoo. A robust image fingerprinting system using the Radon transform. *Signal Processing: Image Communication*, 19(4):325–339, Apr. 2004. ISSN 0923-5965. .

[29] M. Steinebach. Robust Hashing for Efficient Forensic Analysis of Image Sets. In *Digital Forensics and Cyber Crime*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pages 180–187. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-35515-8. .

[30] M. Steinebach, H. Liu, and Y. Yannikos. ForBild: Efficient Robust Image Hashing. In *Media Watermarking, Security and Forensics 2012*, volume 8303, page 83030O. International Society for Optics and Photonics, Feb. 2012. .

[31] M. Steinebach, H. Liu, and Y. Yannikos. Efficient Cropping-Resistant Robust Image Hashing. In *2014 Ninth International Conference on Availability, Reliability and Security*, pages 579–585, Sept. 2014. .

[32] M. Steinebach, S. Lutz, and H. Liu. Privacy and Robust Hashes. In *Proceedings of the 14th International Conference on Availability, Reliability and Security - ARES '19*, pages 1–8, Canterbury, CA, United Kingdom, 2019. ACM Press. ISBN 978-1-4503-7164-3. .

[33] United Nations. Universal Declaration of Human Rights, Dec. 1948. URL http://www.un.org/en/universal-declaration-human-rights/index.html.

[34] Uwe Breidenbach. *Privacy-Enhanced Robust Image Hashing with Bloom Filters*. master, TU Darmstadt, Jan. 2020.

[35] C. Winter, M. Steinebach, and Y. Yannikos. Fast indexing strategies for robust image hashes. *Digital Investigation*, 11:S27–S35, May 2014. ISSN 17422876. .

[36] B. Yang, F. Gu, and X. Niu. Block Mean Value Based Image Perceptual Hashing. In *2006 International Conference on Intelligent Information Hiding and Multimedia*, pages 167–172, Dec. 2006. .

[37] A. C.-C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (Sfcs 1986)*, pages 162–167, Oct. 1986. .

Biographies

Uwe Breidenbach. In 2016 Uwe Breidenbach graduated with a BSc in Computer Science, and in 2020 with a MSc in IT-Security, as well as a

MSc in Computer Science from *Technische Universität Darmstadt*, Germany. He wrote his bachelor and master thesis at the *Fraunhofer Institute for Secure Information Technology*, Darmstadt, Germany. In addition, the master thesis was written in cooperation with the *Faculty of Engineering* of *Mahidol University*, Thailand. His main interest of research and work include penetration testing, ethical hacking, network security, digital forensics, and privacy protection.

Martin Steinebach is the manager of the Media Security and IT Forensics division at Fraunhofer SIT. From 2003 to 2007 he was the manager of the Media Security in IT division at Fraunhofer IPSI. He studied computer science at the Technical University of Darmstadt and finished his diploma thesis on copyright protection for digital audio in 1999. In 2003 he received his PhD at the Technical University of Darmstadt for his work on digital audio watermarking. In 2016 he became honorary professor at the TU Darmstadt. He gives lectures on Multimedia Security as well as Civil Security. He is Principle Investigator at ATHENE and represents IT Forensics and AI Security. Before he was Principle Investigator at CASED with the topics Multimedia Security and IT Forensics. In 2012 his work on robust image hashing for detection of child pornography reached the second rank of the *Deutscher IT Sicherheitspreis*, an award funded by Host Görtz.

Huajian Liu received his B.S. and M.S. degrees in electronic engineering from Dalian University of Technology, China, in 1999 and 2002, respectively, and his Ph.D. degree in computer science from Technical University Darmstadt, Germany, in 2008. He is currently a senior research scientist at Fraunhofer Institute for Secure Information Technology (SIT). His major research interests include information security, digital watermarking, robust hashing and digital forensics.