SOFIR: SECURELY OUTSOURCED FORENSIC IMAGE RECOGNITION

Christoph Bösch *Andreas Peter* *Pieter Hartel* *Willem Jonker*

University of Twente, The Netherlands

ABSTRACT

Forensic image recognition tools are used by law enforcement agencies all over the world to automatically detect illegal images on confiscated equipment. This detection is commonly done with the help of a strictly confidential database consisting of hash values of known illegal images. To detect and mitigate the distribution of illegal images, for instance in network traffic of companies or Internet service providers, it is desirable to outsource the recognition of illegal images to these companies. However, law enforcement agencies want to keep their hash databases secret at all costs as an unwanted release may result in misuse which could ultimately render these databases useless.

We present SOFIR, a tool for the Secure Outsourcing of Forensic Image Recognition allowing companies and law enforcement agencies to jointly detect illegal network traffic at its source, thus facilitating immediate regulatory actions. SOFIR cryptographically hides the hash database from the involved companies. At fixed intervals, SOFIR sends out an encrypted report to the law enforcement agency that only contains the number of found illegal images in the given interval, while otherwise keeping the company's legal network traffic private. Our experimental results show the effectiveness and practicality of our approach in the real-world.

***Index Terms*—** Forensics, law enforcement, network monitoring, somewhat homomorphic encryption.

1 INTRODUCTION

Forensic Image Recognition (FIR) tools are being used by Law Enforcement Agencies (LEAs) worldwide in order to detect illegal images on confiscated equipment. The Dutch police, for example, owns a database consisting of hash values of so-called *PIPs* (Picture that Interests the Police), such as images showing glorification or overexposure of violence, indignity or pornographic content, like zoophilia and pedophilia. When the police confiscates equipment with data storage, the hash of each picture found in the storage is computed and looked up in the PIP database. If there are many matches, the police knows that the confiscated equipment contains PIPs and the investigation is continued manually to crosscheck.

Next to LEAs, companies like Internet service providers (ISPs) or hosting providers, and especially public funded institutions also have an interest in filtering PIPs from their own network traffic. In many countries, ISPs and hosting providers are already filtering out illegal content, either voluntarily [1] or are forced by law (e.g., the Communications Assistance for Law Enforcement Act, CALEA, in the USA). Several companies are even specialized in image filtering techniques for network traffic.

To facilitate the fight of the distribution of PIPs on network traffic, access to the existing police's PIP database would be beneficial. But a major concern of the police when outsourcing the filtering to third parties is the leakage of the PIP database. An even partially disclosed PIP database would allow perpetrators to misuse the database, e.g., by matching their data against the PIP database (before distribution) to check if their images are detectable by the system or not.

A problem with current filter technologies is that they instantly block access to known PIPs. This inherently reveals that the blocked image is in the database, eventually causing the disclosure of the database. Next to the commercial solutions, Peter et al. [2] propose a privacy-preserving architecture to outsource FIR. While preserving the privacy of the owner of the confiscated equipment, their approach unfortunately leaks the PIP database, so its security relies only on legally binding license agreements.

On the other hand, ISPs and especially companies, do not want to expose information on their own network traffic for privacy reasons. Thus the police should learn only the least amount of necessary information to take further legal actions, i.e., the number of actual PIPs detected.

In this paper, we propose SOFIR, a patent-pending [3] Securely Outsourced Forensic Image Recognition tool that inspects network traffic to detect known PIPs. SOFIR allows third parties to scan their network traffic for PIPs, without ever having access to the PIP database. At the same time, the third party reveals only the number of PIPs detected in a certain interval in their network traffic. Our construction is based on an homomorphically encrypted Bloom filter [4, 5].

The rest of the paper is organized as follows: Section 2 introduces the building blocks used in our construction, which in turn is described in Section 3. Our implementation parameters and results are presented in Section 4, while Section 5 concludes with a summary.

2 PRELIMINARIES

We use the following notation and building blocks. Let $\mathcal{DB} = \{d_1, \ldots, d_n\}$ be a database, consisting of n known PIPs.

978-1-4799-2893-4/14/$31.00 ©2014 IEEE

Bloom Filter. A Bloom filter (BF) [6] is a data structure which is used to answer set membership queries. It is represented as an array of m bits which are initially set to 0. We write $\mathrm{B}[i]$ to denote the i-th position of the BF. In general the filter uses k independent hash functions h_j ($1 \leq j \leq k$), where $h_j : \{0,1\}^* \rightarrow [1,m]$ maps a set element to one of the m array positions. For each element e in a set $\mathcal{S} = \{e_1, \ldots, e_n\}$ the bits at positions $\mathrm{B}[h_j(e)]$ are set to 1. To check whether an element x belongs to the set $\mathcal{S}$, we check if the bits at all positions $\mathrm{B}[h_j(x)]$ are set to 1, i.e., if $\prod_{j=1}^{k} B[h_j(x)] \stackrel{?}{=} 1$. If so, x is considered a member of set $\mathcal{S}$. BFs do not produce false negatives, but inherently have a possibility of false positives (FPs), since the positions of an element may have been set by one or more other elements. With appropriate parameters m, n and k, the false positive probability P can be set to a desired low level [7]. The bit size of the BF can be approximated as: $m = 1/\left(1-(1-P^{\frac{1}{k}})^{\frac{1}{kn}}\right)$.

Somewhat Homomorphic Encryption. Somewhat homomorphic encryption (SHE) allows to perform a limited number of different algebraic operations on plaintexts but in the encrypted domain without knowing the decryption key. We use the private-key lattice-based SHE scheme by Brakerski and Vaikuntanathan (BV) [8], which allows for multiplications and additions. Any other probabilistic semantically secure SHE scheme that allows at least one multiplication followed by multiple additions on encrypted values can also be used (e.g., Gentry-Halevi-Vaikuntanathan [9] or Boneh-Goh-Nissim [10]). The homomorphic encryption of an element x is written as $[\![x]\!]$. For the BV scheme we write: $[\![x]\!] \otimes [\![x']\!] = [\![x \otimes x']\!]$, where $\otimes \in \{+, \cdot\}$.

The BV scheme works over polynomials and uses the following parameters: a polynomial degree α (which is a power of 2), a modulus q (which is a prime such that $q \equiv 1 \pmod{2\alpha}$), the cyclotomic polynomial $f(x) = x^\alpha + 1$, the discrete Gaussian error distribution χ with standard deviation σ, the ring $R_q = \mathbb{Z}_q[x]/\langle f(x)\rangle$, the number of supported additions A and multiplications M and a prime $t < q$ which defines the message space as $R_t = \mathbb{Z}_t[x]/\langle f(x)\rangle$.

A freshly generated ciphertext $ct = (c_0, c_1)$ consists of two elements in R_q (i.e., polynomials). We say that ct has *ciphertext degree* $C = 2$. Multiplying two ciphertexts increases the degree of the resulting ciphertext: $(c_0, \ldots, c_a) \cdot (c_0, \ldots, c_b) = (c_0, \ldots, c_{a+b})$. Since each polynomial coefficient is at most of size $q-1$, the ciphertext size $|c| = C \cdot \alpha \cdot \lceil \lg(q) \rceil$ is an upper bound and denoted by WC $|c|$. The security of the scheme is measured by the runtime T of the distinguishing attack [11]. Thus, $\lg(T)$ denotes the bit security of the scheme.

3 OUR SOFIR CONSTRUCTION

Scenario. A Law Enforcement Agency (LEA) encrypts its PIP database and gives it to the ISP (or some other company or hosting provider). The ISP uses the encrypted database to find PIPs in its network traffic and regularly sends an encrypted report on the number of detected PIPs back to the LEA. The LEA can decrypt the report to check the results of the matching and, if necessary, starts an investigation.

Security Requirements. To securely outsource the FIR, we require that the hash values of the database do not leak to anybody. Note that this also includes the protection of the matching result, since this inherently leaks information on the database. To protect the privacy of the ISP, the LEA should learn only the least amount of necessary information possible, i.e., the total number of PIPs found.

Our Construction. We present SOFIR, which consists of three phases: the *initialization phase* (run at the LEA), the *recognition phase* (run at the ISP) and the *revelation phase* (run at the LEA).

During the *initialization phase*, the LEA first generates a secret key K for the BV scheme and initializes a BF. Moreover, an inner hash function h^{in} (to compute the hash value of an image) and several outer hash functions h_j^{out}, for $j \in [1,k]$ (to calculate the BF positions) are chosen.

To insert all PIPs $d \in \mathcal{DB}$ into the BF, first an inner hash value $x = h^{in}(d)$ is computed. Then, for all x, the positions $p_j = h_j^{out}(x)$ for $j \in [1,k]$ are calculated, using the outer hash functions. The BF positions $\mathrm{B}[p_j]$ are set to 1. After all PIPs have been inserted into the BF, it is encrypted bit-by-bit using the BV scheme and the secret key K. The encrypted BF $[\![\mathrm{B}]\!] = ([\![\mathrm{B}[1]]\!], \ldots, [\![\mathrm{B}[m]]\!])$ can then be used in the SOFIR recognition phase by the ISP as we explain momentarily.

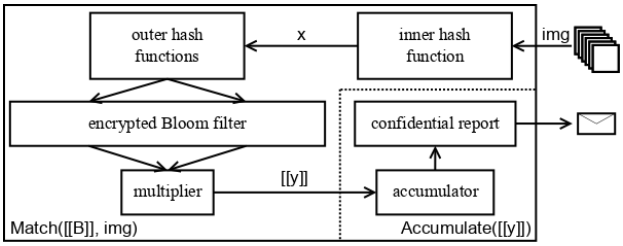

Fig. 1: SOFIR architecture (simplified).

The *recognition phase* (cf. Fig. 1) is split into two algorithms: Match (which identifies PIPs in the encrypted domain) and Accumulate (which adds up all the matching results and sends a confidential report to the LEA). To check the network traffic for PIPs, each image file img is processed in the following way. First, Match uses the inner hash function to calculate $x = h^{in}(\mathsf{img})$. The outer hash function is then used to calculate the BF positions $p_j = h_j^{out}(x)$ for all $j \in [1,k]$. Note that h^{in} and h_j^{out} are the same hash functions as used by the LEA in the initialization phase. The encrypted BF positions $[\![\mathrm{B}[p_j]]\!]$ are processed by the *multiplier*, which uses the multiplicative homomorphic property of the BV scheme to privately compute the matching result $[\![y]\!] = \prod_{j=1}^{k} [\![\mathrm{B}[p_j]]\!]$. The value $[\![y]\!]$ will be $[\![1]\!]$ in case of a match and $[\![0]\!]$ otherwise. The Accumulate algorithm takes $[\![y]\!]$ and adds it (using the additive homomorphic property of

BV) to the final accumulated result $[\![R]\!]$, which is the total number of PIPs matching the database. After a certain time (e.g., one hour or day) or threshold (e.g., 50,000 queries), $[\![R]\!]$ is sent to the LEA and the internal $[\![R]\!]$ is reset to $[\![0]\!]$.

During the *revalation phase*, given $[\![R]\!]$ and the secret key K, the LEA decrypts $[\![R]\!]$ and outputs the number of possible matches. If $R > \tau$, where τ is a certain threshold, an alarm is raised for further investigation.

Security. The security of our construction can be analyzed as follows. Due to the use of a probabilistic semantically secure SHE scheme, it is impossible to distinguish between $[\![0]\!]$ and $[\![1]\!]$. Therefore, the BF itself does not leak any information on the contents. Since all operations for the matching (homomorphic multiplication of the encrypted BF values), as well as the accumulation (homomorphic addition of encrypted values) are performed in the encrypted domain, the ISP cannot gain any information on the encrypted database, the computations or the results. The LEA receives only the accumulated result, which is the number of found PIPs. Thus, the ISP does not reveal information on its network traffic, except the number of detected image files.

4 IMPLEMENTATION AND FEASIBILITY STUDY

This section gives implementation details and a feasibility study, where we explore the parameter space to get realistic numbers for implementing SOFIR.

To setup our system we have to set (i) the system parameters, (ii) BF parameters and finally (iii) BV parameters. We will show experimental results based on a real-world setting.

We start by estimating N, the number of image files that need to be scanned per hour. To get a realistic value, we monitored the network traffic of our University homepage for two weeks and (on average) registered access to around 50,000 image files (i.e., .jpg, .jpeg, .png, .gif) per hour. This is our starting point to determine our parameters.

(i) System Parameters. 50,000 images per hour allows for 71.4 ms (60 min/50,000) of maximal processing time per image and a FP-rate of $1/50{,}000 = 2$ E-05 (one false positive per hour)[1]. Since there is no publicly available information on PIP database sizes, we assume a PIP database consisting of $n = 500{,}000$ PIPs. The size n has an effect only on the BF size m and not on our timing results.

(ii) BF Parameters. For $n = 500{,}000$ we calculate m (cf. Section 2), the bit-size of the BF for different k and FP-rates ($P \leq 2.0$ E-05) as shown in Fig. 2.

Increasing the number of hash functions, significantly decreases the Bloom filter size. We realize the BF lookup by multiplying the corresponding BF values per image ($[\![y]\!] = \prod_{j=1}^{k} [\![\mathrm{B}[p_j]]\!]$). The number of hash functions also determines M, the number of multiplications the BV scheme needs to support ($M = k - 1$). Therefore, we will look at the influence of M on the efficiency of the BV scheme. Recall that

[1]Note that the LEA will post-filter the results to remove false positives in case of an investigation to avoid accusing innocent.

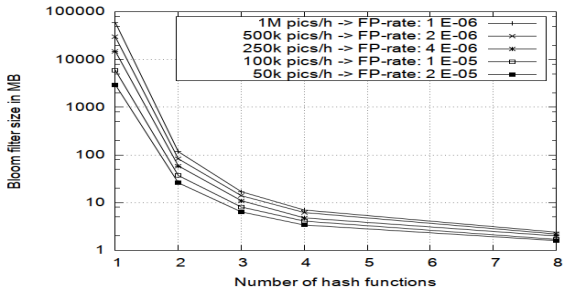

Fig. 2: Bloom filter sizes in MB using $n = 500{,}000$.

we only have 71.4 ms per image.

(iii) BV Parameters. We choose our parameters for the symmetric BV scheme based on the number of images scanned per interval. The accumulator has to perform $A = 50{,}000$ additions. Recall, that the accumulator is adding either an encrypted 0 or 1, implying that 50,000 is the biggest value our encryption scheme needs to be able to handle. Thus, we set the size of the message space $t = 50{,}021$ (next prime $> 50{,}000$).

We also take into account the work of Lauter et al. [12] ($\sigma = 8$) which assessed the security against the decoding attack [11] and the distinguishing attack [13]. With these fixed parameters, we calculate the flexible parameters for different M as seen in Table 1.

	M	α	$\lceil \lg(q) \rceil$	$\lg(T)$	WC $\lvert c \rvert$
(a)	3	4096	140	107	140 kB
(b)	2	4096	100	196	100 kB
(c)	1	2048	77	91	38.5 kB

Table 1: Details of the used BV parameters (cf. Section 2).

Organizing the multiplications in form of a binary tree (cf. Fig. 3) allows us to perform the multiplications in *layers*. Supporting M multiplications (one per layer) in the BV scheme, allows us to use up to $k = 2^M$ hash functions in our BF. We verified the results by experiments.

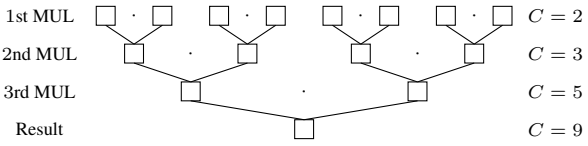

Fig. 3: Multiplication tree for $k = 8$.

We implemented the symmetric BV scheme in C/C++ using FLINT, the Fast Library for Number Theory [14]. We tested the code on an Intel Xeon CPU $X5677$ with 3.47 GHz running linux 3.11.0-sabayon x86_64. Our timing results are shown in Table 2. By using the multiplication tree (cf. Fig. 3), we always multiply ciphertexts with the same degree C. To compute the total processing time, we have to add up the times for all used operations (per layer). For instance, for

(a), we have to compute 4, 2 and 1 multiplication in layers 1, 2 and 3, respectively for the matching, plus the final addition for the accumulator. Thus, the total precessing time per image is 1021 ms $(4 \cdot 63.6 + 2 \cdot 154 + 453 + 5.58)$. Looking at Table 2 we see, that (c) is the only setting, that achieves our required processing time of max. 71.4 ms.

BV	Operation	$C = 2$	$C = 3$	$C = 5$	$C = 9$	Total
(a)	ADD	0.19 ms	1.56 ms	2.94 ms	**5.58** ms	1021 ms
	MUL	**63.6** ms	**154** ms	**453** ms	–	
(b)	ADD	0.82 ms	1.39 ms	**2.6** ms	–	211 ms
	MUL	**48.9** ms	**111** ms	–	–	
(c)	ADD	0.41 ms	**0.65** ms	–	–	20.85 ms
	MUL	**20.2** ms	–	–	–	

Table 2: Implementation results for the BV scheme. Times for a single operation (MUL, ADD) dependent on the ciphertext degree.

Optimizations and Final Results. At this moment, we have only a single threaded implementation of the BV scheme. The BV scheme itself is highly parallelizable and offers several optimization options as mentioned by Lauter et al. [12]. This reduces the times for the homomorphic multiplications and additions from Table 2.

Another possible optimization for SOFIR is to parallelize the image processing and use a single CPU core per image. Modern CPUs consist of 2-48 cores (e.g., AMD Opteron, Intel Xeon). For instance, using a usual 16 core CPU outputs 16 results in 1021 ms for (a), reducing the average processing time to 63.8 ms per image. In this way we achieve our goal of having a processing time per image of less than 71.4 ms. Using the BV parameters (b), a Quad-Core CPU brings the average processing time down to 52.75 ms as shown in Table 3. The final BF size $|[\![\mathrm{B}]\!]|$ is computed as $m \cdot$ WC $|c|$, since each of the m encrypted BF positions is of size WC $|c|$. Recall that $|[\![\mathrm{B}]\!]|$ is an upper bound as explained in Section 2.

Table 3 shows, that depending on the available cores, we have several options to implement the system at our University. Having a CPU with 16 cores, allows us to use the BV parameters (a), resulting in 1.8 TB of storage and a maximum of 56,426 images per hour. A graphical representation of the trade-offs on security, time and storage using different BV parameters for our single-core results from Table 3 is shown in Fig. 4. Note that the timing results do not take the hash functions into account. However, compared to the homomorphic operations the times for hashing are negligible.

Limitations. Like all FIR tools, SOFIR is not able to detect encrypted PIPs. Encryption makes it impossible to access and process the plaintext image without the decryption key. In practice however, most network traffic in companies or at hosting providers is unencrypted. SOFIR is designed to detect illegal images in such (unencrypted) settings. It gives companies more insight into their own network traffic by utilizing the confidential PIP databases held by law enforcement agencies. This is beneficial for both companies and LEAs to detect and mitigate the distribution of PIPs.

M	k	BV	sec.	FP-rate	$\lvert[\![\mathrm{B}]\!]\rvert$	time	time (parallel)
3	8	(a)	107 b	1.5 E-05	1.8 TB	1021 ms	16 core
				1.2 E-06	2.6 TB		63.8 ms
2	4	(b)	196 b	1.7 E-05	2.8 TB	211 ms	4 core
				1.6 E-06	5.1 TB		52.75 ms
1	2	(c)	91 b	1.5 E-05	9 TB	20.2 ms	1 core
				1.6 E-06	28 TB		20.2 ms

Table 3: Final implementation results. Encrypted BF sizes depending on k and BV. Single-core and optimized parallel timings.

Another limitation of our current system is the inability to detect manipulated PIPs. To detect small image manipulations (e.g., cropping, rotating, scaling, shifting, JPEG compression, median filtering), a perceptual/robust hash function should be used in place of the inner hash function h^{in}. Such a perceptual hash function is a compression function that outputs very similar values (in terms of some metric, e.g., the Hamming metric) for perceptually similar pictures. Numerous instantiations using different techniques are known [15, 16]. In its current form, SOFIR is not able to deal with the fuzziness or error-proneness of perceptual hash functions. We consider this as interesting future work.

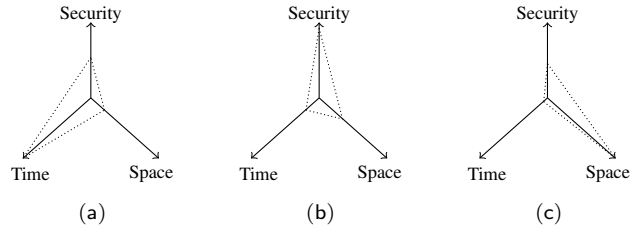

Fig. 4: Graphical representation of the single-core implementation results from Table 3 for different BV parameters.

5 CONCLUSION

We have presented SOFIR, a system to detect known illegal images in network traffic in a privacy-preserving manner. Our mechanism is not limited to images but can also detect all other file formats, e.g., documents or videos. SOFIR cryptographically hides the hash database from the involved companies. The encrypted reports to the LEA only contain the number of found illegal images in a given interval, thereby keeping the company's legal network traffic private. We instantiated our proposed system using the somewhat homomorphic encryption scheme from Brakerski and Vaikuntanathan [8] and showed, that it is efficient to be used in real world application scenarios.

As future work we plan to replace the inner hash function with (i) a perceptual hash funtion to detect small image manipulations and (ii) different feature extraction algorithms, e.g., digital camera identifiers or watermarks, that can identify a camera model or images, respectively, in a unique way.

6 REFERENCES

[1] Internet Watch Foundation, "URL List Recipients," https://www.iwf.org.uk/members/member-policies/url-list/iwf-list-recipients, October 2013.

[2] Andreas Peter, Thomas Hartmann, Sascha Müller, and Stefan Katzenbeisser, "Privacy-Preserving Architecture for Forensic Image Recognition," in *2012 IEEE International Workshop on Information Forensics and Security, WIFS 2012, Costa Adeje, Tenerife, Spain, December 2-5, 2012*. 2012, pp. 79–84, IEEE.

[3] Christoph Bösch, Richard Brinkman, Pieter Hartel, and Willem Jonker, "Method and System of Monitoring a Data Stream," Patent NL2009845 (Application), 2012.

[4] Florian Kerschbaum, "Outsourced private set intersection using homomorphic encryption," in *ASIACCS*, 2012, pp. 85–86.

[5] Henning Perl, Yassene Mohammed, Michael Brenner, and Matthew Smith, "Fast confidential search for biomedical data using Bloom filters and Homomorphic Cryptography," in *eScience*, 2012, pp. 1–8.

[6] Burton H. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[7] Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel H. M. Smid, and Yihui Tang, "On the False-Positive Rate of Bloom Filters," *Inf. Process. Lett.*, vol. 108, no. 4, pp. 210–213, 2008.

[8] Zvika Brakerski and Vinod Vaikuntanathan, "Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages," in *CRYPTO 2011*, Phillip Rogaway, Ed. 2011, vol. 6841 of *Lecture Notes in Computer Science*, pp. 505–524, Springer.

[9] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan, "A Simple BGN-Type Cryptosystem from LWE," in *EUROCRYPT 2010*, Henri Gilbert, Ed. 2010, vol. 6110 of *Lecture Notes in Computer Science*, pp. 506–522, Springer.

[10] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim, "Evaluating 2-DNF Formulas on Ciphertexts," in *TCC 2005*, Joe Kilian, Ed. 2005, vol. 3378 of *Lecture Notes in Computer Science*, pp. 325–341, Springer.

[11] Richard Lindner and Chris Peikert, "Better Key Sizes (and Attacks) for LWE-Based Encryption," in *CT-RSA 2011*, Aggelos Kiayias, Ed. 2011, vol. 6558 of *Lecture Notes in Computer Science*, pp. 319–339, Springer.

[12] Kristin Lauter, Michael Naehrig, and Vinod Vaikuntanathan, "Can Homomorphic Encryption be Practical?," in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, New York, NY, USA, 2011, CCSW '11, pp. 113–124, ACM.

[13] Daniele Micciancio and Oded Regev, *Lattice-Based Cryptography*, pp. 147–191, Springer-Verlag, February 2009.

[14] William Hart, "FLINT: Fast Library for Number Theory," http://www.flintlib.org.

[15] Ramarathnam Venkatesan, S.-M. Koon, Mariusz H. Jakubowski, and Pierre Moulin, "Robust Image Hashing," in *Proceedings of the International Conference on Image Processing, ICIP 2000, September 10-13, Vancouver, BC, Canada*, 2000.

[16] Christoph Zauner, "Implementation and Benchmarking of Perceptual Image Hash Functions," M.S. thesis, Upper Austria University of Applied Sciences, 2010, http://www.phash.org/docs/pubs/thesis_zauner.pdf.