A Secure Submission System for Online Whistleblowing Platforms

Volker Roth, Benjamin Güldenring, Eleanor Rieffel[†], Sven Dietrich,[‡] Lars Ries

Freie Universität Berlin, [†]FX Palo Alto Laboratory,
[‡]Stevens Institute of Technology

Abstract. We motivate and introduce our design and development of a secure submission front-end for online whistleblowing platforms. Our system is designed to provide a level of anonymity in the face of adversaries who can perform end-to-end traffic analysis.

Keywords: Whistleblowing, traffic analysis, homomorphic encryption.

1 Introduction

Corporate or official corruption and malfeasance can be difficult to uncover without information provided by insiders, so-called *whistleblowers.* Even though many countries have enacted, or intend to enact, laws meant to make it safe for whistleblowers to disclose misconduct [2, 12], whistleblowers fear discrimination and retaliatory action regardless, and sometimes justifiably so [4, 10].

It is therefore unsurprising that whistleblowers often prefer to blow the whistle anonymously through other channels than those mandated by whistleblowing legislature. This gave rise to whistleblowing websites such as *Wikileaks.* However, the proliferation of surveillance technology and the retention of Internet protocol data records [3] has a chilling effect on potential whistleblowers. The mere act of connecting to a pertinent Website may suffice to raise suspicion [9], leading to cautionary advice for potential whistleblowers.

The current best practice for online submissions is to use an SSL connection over an anonymizing network such as Tor [7]. This hides the end points of the connection and it protects against malicious exit nodes and Internet Service Providers (ISPs) who may otherwise eavesdrop on or tamper with the connection. However, this does not protect against an adversary who can see most of the traffic in a network [5, 8], such as national intelligence agencies with a global reach and view.

In this paper, we suggest a submission system for online whistleblowing platforms that we call *AdLeaks.* The objective of AdLeaks is to make whistleblower submissions unobservable even if the adversary sees the entire network traffic. A crucial aspect of the AdLeaks design is that it eliminates any signal of intent that could be interpreted as the desire to contact an online whistleblowing platform. AdLeaks is essentially an online advertising network, except that ads carry additional code that encrypts a zero probabilistically with the AdLeaks public key

and sends the ciphertext back to AdLeaks. A whistleblower's browser substitutes the ciphertext with encrypted parts of a disclosure. The protocol ensures that an adversary who can eavesdrop on the network communication cannot distinguish between the transmissions of regular browsers and those of whistleblowers' browsers. Ads are digitally signed so that a whistleblower's browser can tell them apart from maliciously injected code. Since ads are ubiquitous and there is no opt-in, whistleblowers never have to navigate to a particular site to communicate with AdLeaks and they remain unobservable. Nodes in the AdLeaks network reduce the resulting traffic by means of an *aggregation* process. We designed the aggregation scheme so that a small number of trusted nodes with access to the decryption keys can recover whistleblowers' submissions with high probability from the aggregated traffic. Since neither transmissions nor the network structure of AdLeaks bear information on who a whistleblower is, the AdLeaks submission system is immune to passive adversaries who have a complete view of the network.

In what follows, we detail our threat model and our assumptions, we give an overview over the design of AdLeaks, we report on the current state of its implementation, we summarize the outcome of our scalability analysis, and we explain how AdLeaks uses cryptographic algorithms to achieve its security objectives. We give a more detailed report in [14].

2 Assumptions and Threats in Our Scope

The primary security objective of AdLeaks is to conceal the *presence* of whistleblowers, and to eliminate network traces that may make one suspect more likely than another in a search for a whistleblower. We assume that whistleblowers use AdLeaks only on private machines to which employers have no access. In fact, sending information from work computers even using work-related e-mail accounts is a mistake whistleblowers make frequently. We hope that the software distribution channels we discuss in Section 3.3 will help reminding whistleblowers to not make that mistake.

AdLeaks addresses the threat of an adversary who has a global view of the network and the capacity to store or obtain Internet protocol data records for most communications. The adversary may even require anonymity services to retain connection detail records for some time and to provide them on request. The adversary may additionally store selected Internet traffic and he may attempt to mark or modify communicated data. However, we assume that the adversary has no control over users' end hosts, and he does not block Internet traffic or seizes computer equipment without a court order. We assume that the court does not *per se* consider organizations that relay secrets between whistleblowers and journalists as criminal. The objective of the adversary is to uncover the identities of whistleblowers. The threat model we portrayed is an extension of [3] and it is likely already a reality in many modern states, or it is about to become a reality. For reasons we explain in [14] we do not consider additional threats that we would doubtless encounter, for example, in technologically advanced totalitarian countries.

3 System Architecture

AdLeaks consists of two major components. The first component is an online advertising network comparable to existing ones. The network has advertising partners (the publishers) who include links or scripts in their web pages which request ads from the AdLeaks network and display them. Publishers may receive compensation in accordance with common advertising models, for example, per mille impressions, per click or per lead generated. Advertisers run campaigns through the AdLeaks network. AdLeaks may additionally run campaigns through other ad networks to extend its reach, for example, funded by donations or profits from its own operations. The ecosystem of partners and supporters may include large newspapers, bloggers, human rights organizations and their affiliates. For example, Wikileaks has partnered with organizations such as Der Spiegel, El País and the New York Times, and OpenLeaks had hinted at support by Greenpeace and other organizations. The key ingredient of an AdLeaks ad is not its visual display but its active JavaScript content. Supporters who would forfeit significant revenue when allocating advertising space to AdLeaks ads have a choice to only embed the JavaScript portion. The JavaScript is digitally signed by AdLeaks and contains public encryption keys.

The second major component of AdLeaks is its submission infrastructure. This infrastructure consists of three tiers of servers. We refer to these tiers as *guards, aggregators* and *decryptors.* When a browser loads an AdLeaks ad, the embedded JavaScript encrypts a zero probabilistically with the embedded public key and submits the ciphertext to a guard. The guard strips unnecessary encoding and protocol meta-data from the request and forwards the ciphertext to an aggregator. An aggregator aggregates the ciphertexts it receives per second and transmits them to the decryptor. What makes this setting challenging is that we want to limit the bandwidth of the decryptor to a household Internet connection so that we can keep a close eye on the all-important machine with the decryption keys. The aggregation leverages the homomorphic properties of the Damgård-Jurik (DJ) encryption scheme [6], which means that the product of the ciphertexts is an encryption of the sum of the plaintexts. We chose the DJ scheme because it has a favorable plaintext to ciphertext ratio.

The decryptor decrypts the downloaded ciphertexts and, if it finds data in them, reassembles the data into files. The files come from whistleblowers. In order to submit a file, a whistleblower must first *obtain* an installer that is digitally signed and distributed by AdLeaks. This is already a sensitive process that signals intent. We defer the discussion of safe distribution channels for the installer to section 3.3. *Installing* the obtained software likewise signals the intent to disclose a secret, and therefore it is crucial that the whistleblower verifies the signature *before* running the installer, and assures himself that the signer is indeed AdLeaks. Otherwise, he is vulnerable to Trojan Horse software designed to implicate whistleblowers. When run, the installer produces an instrumented browser and an encryption tool. The whistleblower prepares a file for submission by running the encryption tool on it. The tool's output is a sequence of ℓ ciphertexts. Henceforth, whenever an instrumented browser runs an ad signed

by AdLeaks, it replaces the script's ciphertext with one of the ℓ ciphertexts it has not already used as a replacement.

In order to distinguish ciphertexts that are encryptions of zeros from ciphertexts that are encryptions of data we refer to the former as *white* and to the latter as *gray*. If the aggregator aggregates a set of white ciphertexts then the outcome is another white one. If exactly one gray ciphertext is aggregated with only white ones then the outcome is gray as well. If we decrypt the outcome then we either recover the data or we determine that there was no data to begin with. If two or more gray ciphertexts are aggregated then we cannot recover the original data from the decryption. We call this event a *collision* and we refer to such an outcome as a *black* one. Obviously, we must expect and cope with collisions in our system. In what follows, we elaborate on details of the design that are necessary to turn the general idea into a feasible and scalable system.

3.1 Disclosure Preparation

In order to handle collisions, the encryption tool breaks a file into blocks of a fixed equal size and encodes them with a loss tolerant *Fountain Code*. Fountain codes encode n packets into an infinite sequence of output packets of the same size such that the original packets can be recovered from any n' of them where n' is only slightly larger than n. For example, a random linear Fountain Code decodes the original packets with probability $1 - \delta$ from about $n + \log_2(1/\delta)$ output packets [11]. Let n'' be somewhat larger than n' and let $m_1, \ldots, m_{n''}$ be the Fountain encoding of the file. The tool then generates a random file identification number k and computes: $c_i = \mathrm{Enc}^{\mathrm{cca}}_{\kappa_1}(\mathrm{EncData}_{\kappa_2}(m_i, k||i||n))$ for $1 \leq i \leq n''$ where κ_1 is an aggregator key and κ_2 is the actual submission key. The purpose of the dual encryption will become clear in Section 4. We assume that the outer encryption is a fast hybrid IND-CCA secure cipher. We defer the specification of the inner encryption scheme to Section 6. It assures that, when the decryptor receives the ciphertexts, it can verify the integrity of individual chunks and of the message as a whole and he can associate the chunks that belong to the same submission with all but negligible probability (in $|k|$).

3.2 Decryption

It is substantially cheaper to multiply two DJ ciphertexts in the ciphertext group than it is to decrypt one. Furthermore, the product of ciphertexts decrypts to the sum of the plaintexts in the plaintext group. Recollect that we expect to receive a large number of white ciphertexts, that is, encryptions of zeroes. This leads to the following optimization: we form a full binary tree of fixed height, initialize its leaves with received ciphertexts $c_1, \ldots, c_n$ and initialize each inner node with the product of its children. Then, we begin to decrypt at the root. If the plaintext is zero then we are done with this tree, because all nodes in the tree are zeroes. Otherwise, the decryption yields $\gamma = \alpha + \beta \neq 0$ where α, β are the plaintexts of the left and right child, respectively. We decrypt the left child, which yields α, and calculate the plaintext of the right child as $\beta = \gamma - \alpha$ (without explicit

decryption). If α or β are zeroes then we ignore the corresponding subtree. Otherwise, we recurse into the subtrees that have non-zero roots. If a node is a leaf then we decrypt and verify it. If we find it invalid then we ignore the leaf. Otherwise we forward its plaintext to the file reassembly process. This algorithm saves us 61% decryptions or more, depending on the system load [14]. Our analysis and measurements suggest that a 12-core Mac Pro can serve 51480 concurrent whistleblowers at any time with a 18 Mb/s uplink for the decryptor, independent of the number of users whom AdLeaks serves ads.

3.3 Software Dissemination

We cannot simply offer the installer software for download because the adversary would be able to observe that. Instead, we pursue a multifaceted approach to software distribution. Our simplest and preferred approach involves the help of partners in the print media business. At the time of writing, popular print media often come with attached CDROMs or DVDs that are loaded with, for example, promotional material, games, films or video documentaries. Our installer software can be bundled with these media. Our second approach is to encode the installer into a number of segments using a Fountain Code. In this approach, AdLeaks ads randomly request a segment that the browser loads into the cache. A small bootstrapper program extracts the segments from the browser cache and decodes the installer from it when enough of them have been obtained. Since extraction happens outside the browser it cannot be observed from within the browser. The bootstrapper can be distributed in the same fashion. This reduces the distribution problem to extracting a specific small file from the cache, for example, by searching for a file with a specific signature or name in the cache directory. This task can probably be automated for most platforms with a few lines of script code. The code can be published periodically by trusted media partners in print or verbatim in webpages or it could even be printed on T-Shirts. Our third approach is to enlist partners who bundle the bootstrapper with distributions of popular software packages so that many users obtain it along with their regular software. With our multifaceted approach we hope to make our client software available to most potential whistleblowers in a completely innocuous and unobservable fashion.

4 Security Properties

Eavesdropping and Traffic Analysis AdLeaks funnels all incoming transmissions to the decryptor, and transmissions occur without any explicit user interaction. Hence, the posterior probability that anyone is a whistleblower, given his transmission is observed anywhere in the AdLeaks system, equals his prior probability. From that perspective, AdLeaks is immune against adversaries who have a complete view of the network. Furthermore, AdLeaks' deployment model is suitable to leapfrog the long-drawn-out deployment phase of anonymity systems that rely on explicit adoption. For example, if *Wikipedia* deployed an AdLeaks script then

AdLeaks would reach 10% of the Internet user population overnight, based on traffic statistics by *Alexa* [1].

Outer Encryption and Dishonest Aggregators Assume that AdLeaks did not use outer encryption. Then adversaries might employ the following *active* strategy to gain information on who is sending data to AdLeaks. The adversary samples ciphertexts of suspects from the network and aggregates the ciphertexts for each suspect. He prepares a genuine-looking disclosure that is enticing enough so that the AdLeaks editors will want to publish it with high priority. We call this disclosure the *bait.* The adversary then aggregates suspects' ciphertexts to his disclosure and submits it. If AdLeaks does *not* publish the bait within a reasonable time interval then the adversary concludes that the suspect is a whistleblower. The reasoning is as follows. If the suspect ciphertexts were zeroes then the bait is received and likely published. Since the bait was not published, the suspect ciphertexts carried data which invalidated the bait ciphertexts. This idea can be generalized to an adaptive and equally effective non-adaptive attack that identifies a single whistleblower in a group of W suspects at the expense of $\log_2 W$ baits. For this reason, AdLeaks employs an outer encryption which prevents this attack. However, if an adversary takes over an aggregator then he is again able to launch this attack. Therefore, aggregators should be checked regularly, remote attestation should be employed to make sure that aggregators boot the correct code, and keys should be rolled over regularly. Note that it may take months before a disclosure is published and that a convincing bait has a price — the adversary must leak a sufficiently attractive secret in order to make sure it is published. From this, the adversary only learns that a suspect has sent something but not what was sent.

5 Implementation

We developed fully-functional multi-threaded aggregation and decryption servers with tree decryption support as well as a Fountain Code encoder and decoder. Decryptors write recovered data to disk and the decoder recovers the original file. We also developed a fake guard server which is capable of generating and sending chunks according to a configurable ratio of white and gray ciphertexts. All servers connect to each other through SSH tunnels via port forwarding. The entire implementation consists of 101 C, header and CMake files with 7493 lines of code overall. This includes our optimized DJ implementation [6], which is based on a library by Andreas Steffen, a SHA-256 implementation by Olivier Gay, and several benchmarking tools. Our ads implement the DJ scheme based on the *JSBN.js* library and use *Web Workers* to isolate the code from the rest of the browser. The entire ad currently measures less than 81 KB. The size can be reduced further by eliminating unused library code and by compressing it. The ad submits ciphertexts via *XmlHttpRequests.* We instrumented the Firefox browser for our prototype and patched the source code in two locations. First, we hook the compilation of *Web Worker* scripts and tag every script as an AdLeaks

script if it is labeled as one in lieu of carrying a valid signature. We placed a second hook where Firefox implements the *XmlHttpRequest.* Whenever the calling script is an AdLeaks script running within a *Web Worker,* we replace the zero chunk in its request with a data chunk. Since Web Workers run concurrently the cryptographic operations do not negatively affect the browsing experience.

6 Ciphertext Aggregation Scheme

Our ciphertext aggregation scheme is based on the Damgård-Jurik (DJ) scheme, which IND-CPA secure and is also an isomorphism of

$$\psi_s : \mathbb{Z}_{N^s} \times \mathbb{Z}_N^* \leftrightarrow \mathbb{Z}_{N^{s+1}}^* \qquad \psi_s(a; b) \mapsto (1+N)^a \cdot b^{N^s} \bmod N^{s+1}$$

where N is a suitable public key. The parameter s controls the ratio of plaintext size and ciphertext size. We use two two DJ encryptions c, t to which we jointly refer as a *ciphertext.* We refer to t separately as the *tag.* The motivation for this arrangement is improved performance. We wish to encrypt long plaintexts and the costs of cryptographic operations increase quickly for growing s. Therefore we split the ciphertext into two components. We use a shorter component with $s = 1$, which allows us to test quickly whether the ciphertext encrypts data or a zero. The actual data is encrypted with a longer component with $s > 1$. The two components are glued together using Pederson's commitment scheme [13], which is computationally binding and perfectly hiding. This requires two additions to the public key, which are a generator g of the quadratic residues of $\mathbb{Z}_N^*$ and some $h = g^x$ for a secret x. Instead of committing to a plaintext the sender commits to the hash of the plaintext and some randomness. We use a collision resistant hash function H for this purpose, which outputs bit strings of length $|N/16|$. Furthermore, let R be a source of random bits. The details of the data encryption and decryption algorithms are as follows:

EncData(m, r_0) =
  $r_1, r_2 \leftarrow R$
  chk $\leftarrow$ if $m, r_0 = 0$ then 0
    else $H(m, r_0)$
  $c \leftarrow \psi(m; h^{\text{chk}} \cdot g^{r_1}))$
  $t \leftarrow \psi(r_0 || r_1; g^{r_2})$
  return c, t

DecVrfy(c, t) =
  $(m; k), (r_0 || r_1; \cdot) \leftarrow \psi^{-1}(c), \psi^{-1}(t)$
  chk $\leftarrow$ if $m, r_0 = 0$ then 0
    else $H(m, r_0)$
  if $h^{\text{chk}} \cdot g^{r_1} = k$ then
    return m, r_0
  return $\perp$

We assume that $|r_0|, |r_1|, |r_2|$ are polynomial in the security parameter. Here, r_0 corresponds to $k||i||n$ as we introduced it in Section 3.1. We define EncZero = EncData$(0, 0)$. Aggregation is simply the multiplication of the respective ciphertext components. In order to avoid fields overflowing into adjacent ones we assume that r_0, r_1, r_0', r_1' are left-padded with zeroes. The amount of padding determines how many ciphertexts we can aggregate in this fashion before an additive field overflows into an adjacent one and corrupts the ciphertext. If we use B bits of padding then we can safely aggregate up to 2^B ciphertexts. A length of $B = 40$ is enough for our purposes.

7 Conclusions

AdLeaks leverages the ubiquity of online advertising to provide anonymity and unobservability to whistleblowers making a disclosure online. The system introduces a large amount of cover traffic in which to hide whistleblower submissions, and aggregation protocols that enable the system to manage the huge amount of traffic involved, enabling a small number of trusted nodes with access to the decryption keys to recover whistleblowers' submissions with high probability. We analyzed the performance characteristics of our system extensively, please refer to [14] for details. Our research prototype demonstrates the feasibility of such a system. We expect many aspects of the system can be improved and optimized, providing ample opportunity for further research.

Acknowledgements: The first, second and last author are supported by an endowment of *Bundesdruckerei GmbH.*

References

1. Alexa. Online at `http://www.alexa.com`, Apr. 2012.
2. D. Banisar. *Whistleblowing — International Standards and Developments.* Transparency International, Feb. 2009.
3. S. Berthold, R. Böhme, and S. Köpsell. Data retention and anonymity services. In *The Future of Identity in the Information Society*, volume 298 of *IFIP Advances in Information and Communication Technology*, pages 92–106. Springer, 2009.
4. E. R. Center. 2011 National Business Ethics Survey. Online at `http://www.ethics.org/nbes`, 2345 Crystal Drive, Suite 201, Arlington, VA 22202, USA, 2012.
5. S. Chakravarty, A. Stavrou, and A. D. Keromytis. Traffic analysis against low-latency anonymity networks using available bandwidth estimation. In *Proc. ESORICS*, pages 249–267. Springer, 2010.
6. I. Damgård, M. Jurik, and J. Nielsen. A generalization of Paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9:371–385, 2010.
7. R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *Proc. USENIX Security Symposium*, pages 303–320, 2004.
8. K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *IEEE Symposium on Security and Privacy*, pages 332–346, 2012.
9. S. Gustin. Columbia university reverses anti-WikiLeaks guidance. `http://www.wired.com/threatlevel/2010/12/columbia-wikileaks-policy/`, Dec. 2010.
10. K. J. Lennane. "Whisteblowing": a health issue. *British Medical Journal*, 307:667–670, Sept. 1993.
11. D. MacKay. Fountain codes. *IEE Proceedings*, 152(6), Dec. 2005.
12. A. Osterhaus and C. Fagan. *Alternative to Silence — Whistleblower Protection in 10 European Countries.* Transparency International, 2009.
13. T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proc. CRYPTO*, volume 576, pages 129–140. Springer, 1992.
14. V. Roth, B. Güldenring, E. Rieffel, S. Dietrich, and L. Ries. A secure submission system for online whistleblowing platforms. Online at `http://arxiv.org/abs/1301.6263`, Jan. 2013.