CS CAPSTONE TECHNOLOGY REVIEW

NOVEMBER 20, 2017

PRIVACY PRESERVING CLOUD, EMAIL, AND PASSWORD SYSTEMS

PREPARED FOR

OSU

ATTILA YAVUZ

PREPARED BY

GROUP 38

THE SECRET BUNNY TEAM

ANDREW EKSTEDT

Abstract

We review technical choices for the Privacy Cloud project, including research into searchable encryption algorithms, choices for encryption schemes, and choices for cloud storage providers.

Contents

1 Introduction

The purpose of this document is to outline the technical choices for our project, Privacy Preserving Cloud Encryption. The goal of the project is to implement a certain searchable encryption algorithm and investigate how it can be integrated with common internet applications.

2 Searchable Encryption

2.1 Overview

At the heart of our project is the searchable symmetric encryption algorithm (SSE). Our goal is to implement the scheme described by David Cash et al. in [6]. As part of this tech review we have made an annotated bibliography of some papers about searchable encryption: [15] [6] [14]

3 Encryption scheme

3.1 Overview

The SSE we will implement requires an encryption scheme which has *pseudorandom ciphertexts under chosen plaintext attacks* (RCPA-secure). This is a slightly weaker(?) property than the standard notion of CPA security, so any CPA-secure cipher should suffice. All modern ciphers are CPA-secure.

3.2 Criteria

1) **Security**. The encryption scheme should be secure; that is, there should be no known practical attacks against it which are significantly faster than brute force.
2) **Speed**. The encryption scheme should have performance on par with modern schemes. The faster the better, since we expect encryption operations may be a bottleneck in the algorithm.
3) **Availability**. The encryption scheme should be available in commonly available crypto libraries such as OpenSSL, Crypto++, or tomcrypt.

3.3 Potential choices

3.3.1 AES-CTR

AES (Advanced Encryption Standard) is an NIST standard block cipher, standardized in 2001. CTR (Counter) mode is a standard cipher mode which allows a block cipher to encrypt messages of arbitrary length. CTR mode is secure if the underlying block cipher is secure.

Security: Considered secure. The best published attack against AES-128 takes about 2^{126} steps, which is just barely faster than the brute force time complexity of 2^{128}. This theoretical attack does not represent a meaningful threat in practice.

Speed: AES can be implemented efficiently in software. Recent Intel CPUs have hardware instructions for AES that are even faster.

Availability: OpenSSL, Crypto++, and tomcrypt all support AES-CTR.

Client preference: our client has suggested that we use AES-CTR.

3.3.2 ChaCha20

ChaCha20 [4] is a stream cipher. It was designed by Daniel Bernstein as an extension of his earlier cipher Salsa20, which was the winner of the eSTREAM stream cipher contest. ChaCha20 (and Salsa20) were designed to be very fast in software, and be simple to implement. ChaCha20 is not as widely known as other ciphers, so it may not have as good library support, and it has not been as closely scrutinized as AES, but has been seeing increasing use as an alternative to AES, especially in mobile devices. In 2016, ChaCha20-Poly1305 (ChaCha20 encryption plus Poly1305 authentication) was added to TLS as an official cipher suite. [13]

Speed: ChaCha20 is one of the faster ciphers around, even outpacing AES on systems without hardware AES support. [5]

Security: ChaCha20 is believed to be secure. The best known attack breaks only 7 out of 20 rounds of the cipher. There are no known attacks on the full 20-round cipher better than brute force.

Availability: tomcrypt and Crypto++ have support for ChaCha20. OpenSSL added ChaCha20 in version 1.1.0 (August 2016), which is recent enough that many linux distros may not have picked it up yet.

3.3.3 3DES

DES (Data Encryption Standard) was the standard block cipher before AES. It is the oldest of the three ciphers we have considered, and as such is likely has the widest support. On the other hand, it is also considered to be thoroughly broken because its small key size makes it vulnerable to practical brute force attacks. DES only survives in the present day in the from of Triple DES (3DES), which encrypts data thrice with three separate keys, effectively doubling the key size. The downside is that 3DES is very slow.

Security: Broken.

Speed: Slow.

Availability: Widely supported.

3.4 Conclusion

We choose AES-CTR because it offers the best balance between security, speed, and availability.

We reject ChaCha20 because it does not offer any major advantages over AES and has slightly worse library support.

We reject Triple DES because it is much slower and less secure than AES.

4 Cloud storage

4.1 Overview

While our initial SSE implementation will use a client-server model in which both the client and the server perform steps of the SSE algorithm, we are interested in whether it is possible to do SSE with a ”dumb server” that is only capable of providing storage, not computation.

4.2 Criteria

1) **Cost**. For ease of testing, we would like to be able to use the service for free or for a low cost.
2) **API / Client libraries**. The service must provide some sort of API which we can use to access and upload data. An HTTP API would suffice, but it would be nice if they also provide client libraries for a wide variety of common

languages. In particular, we would like a C++ library because C++ will likely be the primary language of our project.

3) **Popularity**. The aim of this project is to show how SSE can be integrated with services that people actually use. Accordingly, it would be better to use a popular service than an obscure one.

4.3 Potential choices

4.3.1 Google Drive

Google Drive is Google's cloud storage solution. Aside from being able to store arbitrary files, it also integrates with Google's suite of office applications like Google Docs and Google Slides, which we have no particular need for.

Cost: OSU provides students free Google Drive accounts with unlimited storage space.

API: There is an HTTP API. And, although not advertised in the primary documentation, there is a C++ library. [10] Other languages with official library support include: Python, Java, JavaScript, .NET, Obj-C, and PHP. [11]

Popular: Yes. 240 million users as of 2014. [12]

4.3.2 Dropbox

Dropbox [8] is one of the oldest cloud storage services still in existence. They are primarily known for the ability to sync a folder between all your computers.

Cost: Dropbox offers a free tier which provides 2GB of storage capacity. [7]

API: Dropbox has an HTTP API, [9] but no C++ client library. Available client libraries include: Swift, Objective-C, Python, .NET, Java, and JavaScript.

Popular: Yes. 300 million users as of 2014. [12]

4.3.3 Amazon S3

Amazon S3 [3] is a data storage service offered by Amazon. Unlike the previous two options, it is aimed at developers, not consumers. It integrates well with Amazon EC2, which we are considering using for the server side of our project.

Cost: Amazon offers 5GB of storage and 15GB/month of data transfer for free with AWS Free Tier. [2] Additionally, transfer of data between S3 and EC2 is free, which might be nice if we use EC2 for our server hosting.

API: There is an HTTP API, and an official C++ library. [1] Other supported languages include: Python, Java, .NET, Node.js, PHP, Ruby, and Go.

Popular: Yes.

Client preference: our client has requested that we use S3.

4.4 Conclusion

Any one of the three choices would be adequate.

We choose S3 because it provides what we need for free, it has a C++ library available, and our client has requested it. Google Drive would also be a good choice.

We reject Dropbox because they offer the least amount of free storage of the 3 options, and because they do not offer a C++ library.

References

[1] "AWS SDK for C++." [Online]. Available: https://github.com/aws/aws-sdk-cpp

[2] Amazon, "Cloud Storage Pricing Amazon Simple Storage Service (S3) AWS." [Online]. Available: https://aws.amazon.com/s3/pricing/

[3] ——, "Amazon Simple Storage Service (S3) - Cloud Storage - AWS," 2017. [Online]. Available: https://aws.amazon.com/s3/

[4] D. J. Bernstein., "Chacha, a variant of salsa20," Jan. 2008. [Online]. Available: http://cr.yp.to/papers.html#chacha

[5] D. J. Bernstein and T. Lange, "eBACS: ECRYPT Benchmarking of Cryptographic Systems," accessed 2017-11-19. [Online]. Available: https://bench.cr.yp.to

[6] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rou, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation," Cryptology ePrint Archive, Report 2014/853, 2014. [Online]. Available: https://eprint.iacr.org/2014/853

The authors describe a dynamic symmetric searchable encryption scheme in which the encrypted index is stored as a hash table. The keys derived from the search token, and the values are the encrypted file index. Collisions are avoided by running a per-token counter through a MAC — in essence, using an open addressing hash table. The main search index is actually static — updates are performed by storing added keywords in a separate hash table of added keywords, and deleted keywords in a separate set of deleted keywords. The search index should be periodically reconstructed to take these modifications into account. They also describe several variants of the basic scheme which have better methods for reducing the size of the state. Search time is $\mathcal{O}(r)$ (which is optimal), update time is not given, and index size is $\mathcal{O}(N)$, where r is the number of files which match a given query and N is the number of keyword-file pairs in the index. The algorithm is parallelizable.

[7] Dropbox, "Choose the right Dropbox for you and your business," 2017. [Online]. Available: https://www.dropbox.com/plans

[8] ——, "Dropbox," 2017. [Online]. Available: https://www.dropbox.com/

[9] ——, "Dropbox - Developers," 2017. [Online]. Available: https://www.dropbox.com/developers/documentation

[10] Google, "Google APIs Client Library for C++," 2017. [Online]. Available: https://google.github.io/google-api-cpp-client/latest/

[11] ——, "Google Drive Platform: Downloads — Drive REST API — Google Developers," 2017. [Online]. Available: https://developers.google.com/drive/v3/web/downloads

[12] E. Griffith, "Who's winning the consumer cloud storage wars?" [Online]. Available: http://fortune.com/2014/11/06/dropbox-google-drive-microsoft-onedrive/

[13] A. Langley, W.-T. Chang, N. Mavrogiannopoulos, J. Strombergson, and S. Josefsson, "ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS)," RFC 7905, June 2016. [Online]. Available: https://rfc-editor.org/rfc/rfc7905.txt

[14] D. X. Song, D. Wagner, S. David, and A. Perrig, "Practical techniques for searches on encrypted data," 2000.

The original searchable encryption paper. The authors introduce the concept of searchable encryption and define the essential properties of a searchable encryption scheme: that the server should not be able to learn anything about the plaintext; that the server should not be able to learn which keywords the user is searching for; and that the server cannot perform a search without the user's permission. They do not rigorously define what operations a searchable encryption scheme supports. They mention dynamic schemes only in passing. They also present a basic scheme and prove its security. Remarkably, the scheme they present does not use a search index(!); rather, it performs a linear scan over the encrypted documents. They briefly discuss a scheme which uses an index at the end, but do not go into detail. Basically, the scheme works by encrypting the list of tokens and xoring each encrypted token with a pair of values (L,R) such that L and R are related, but in a way that can only be detected if the server knows the keyword. To search, the client provides the server with the encrypted keyword and the server performs a linear search over the encrypted document, xoring each block with the encrypted keyword and checking if the property holds. Notably, this search method is probabilistic(!) because it can have false positives. The scheme leaks information about where tokens appear in files. Search runs in $\mathcal{O}(n)$ time, where n is the *size of all the documents to be searched*. The paper claims that this is efficient.

[15] A. A. Yavuz and J. Guajardo, "Dynamic searchable symmetric encryption with minimal leakage and efficient updates on commodity hardware," Selected Areas in Cryptography (SAC) 2015, Sackville, New Brunswick, Canada, August 2015, http://web.engr.oregonstate.edu/~yavuza/Yavuz_DSSE_SAC2015.pdf.

The authors describe a dynamic searchable encryption scheme in which the search index is stored as an adjacency matrix where the rows correspond to keywords. Each row is encrypted with a stream cipher using a per-row key. Updates are performed by replacing the relevant column. They claim better security than all previous schemes and competitive performance. Search is $\mathcal{O}(m)$, update is $\mathcal{O}(m)$, server state size is $\mathcal{O}(mn)$, and client state size is $\mathcal{O}(n + m)$ where m is the number of keywords and n is the number of files. The algorithm is parallelizable.