leviathan
security group

CRYPTON SECURITY AUDIT

FINAL REPORT

MARCH 24, 2014

Leviathan Security Group
3220 1st Ave. S., #100
Seattle, WA 98134

Table of Contents

Crypton Design and Implementation Evaluation

Updated 04/14/14: A previous version of this report contained a non-security finding. As the finding had no bearing on the security of the system, it has since been removed.

Summary

Leviathan recently took a critical look at a technology currently in development by SpiderOak called Crypton. Crypton, an open-source project hosted on github, aims to be a zero-knowledge, cryptographically-secure storage framework upon which zero-knowledge cloud applications can be written. Its primary functional goal is to provide assurance that data stored on the server can only be read by the client who possesses the key and cannot be read by the server or anyone else.

For the version Leviathan reviewed, Crypton was still in development, and we would characterize its readiness as "pre-alpha." As we write this, major components are still being written while others are being rewritten. With that in mind, we were tasked with reviewing two parts of the Crypton framework: a feature called "Sharing" and Crypton's implementation of the Secure Remote Password (SRP) protocol. Since time allowed, we also reviewed the non-SRP login procedures.

Crypton consists of a server and a client, both of which are written in Javascript. The server code, running on node-js, accepts requests from the client and stores or retrieves the appropriate information from a database. The client does most of the heavy lifting; it encrypts, decrypts, signs, and verifies all the user-supplied data. In its current form the client code is pushed to the browser from the server. SpiderOak's true implementation target is for the client code to be in a browser extension or mobile app. An unusual part of Crypton's threat model is that server itself is considered untrusted. That is to say, in all scenarios we anticipated that the server could be acting against the clients. As such, the clients should not trust anything the server gives them. This attribute of the system strongly shaped our assessment.

Secure Remote Password (SRP)

SRP, defined by RFC 2945, is an authentication protocol whereby clients do not reveal their password to the server. SRP is also resistant to replay and man-in-the-middle attacks. We encourage readers to review the brief RFC to understand the workings of the protocol. Unfortunately, SRP is not designed to resist modern password cracking attacks – the Verifier[1], as defined by the RFC, is merely two rounds of SHA1 plus a modular exponentiation; we would not describe this as robust against modern cracking techniques and tools. As a result, when we consider the scenario for Crypton where the server is malicious, the server itself could execute password cracking attacks against the verifier to gain decryption keys to a user's information. While Crypton's implementation used SHA256 instead of SHA1,

[1] The Verifier (v) is defined as:
x = SHA(<salt> | SHA(<username> | ":" | <raw password>))
v = g^x % N

we recommended use of bcrypt as the hash function. Bcrypt is well-known for being computationally intensive, which results in improved password cracking resistance.

Primary Security Issues

With an understanding of what Crypton intends to be and generally how it works, we examined it with a critical eye and observed places where things could go wrong. Upon successful login, the server sends the client its encrypted account information. The client then generates its master decryption key and begins decrypting its account information. Examining this, we observed three key issues:

- First, the account's public key is stored without verification that it corresponds to the decrypted private key; this could allow the server to replace it which would result in the user encrypting messages to themselves that they cannot actually decrypt.
- Second, we saw the same issue with the account's public signing key; it is not being verified against the decrypted private signing key.
- Third, the containerNameHmacKey is not verified before decryption which means that the server could actually replace it with a different one that it knows by encrypting a new symmetric key to the user's public key.

Crypton's Unsolved Problem

Every user-to-user encrypted messaging platform (including OTR, Cryptocat, and TextSecure) has authentication limitations. Users are not fully sure that they are talking to the intended party. Crypton is no different – since the server cannot be trusted, a given client cannot be sure the server is sending the correct public key when they first interact with another user. Each of the aforementioned platforms handle this problem in its own way, and we look forward to seeing what kind of innovative method Crypton implements to handle this.

Takeaways

Crypton is an ambitious and novel platform. To our knowledge, no one else has set out to create a similar framework for building future secure cloud applications. Based on the state of Crypton at the time of Leviathan's review, the groundwork seemed to be nearly complete, and the development team stated that it is working on significant improvements to the developer's experience. In particular, the team has already figured out how to index encrypted data to allow for searchable diary entries. They are in the process of creating higher-level libraries to handle complex data structures and algorithms, all implemented atop the encrypted block storage provided by Crypton.

Throughout the review we had an ongoing conversation with Crypton's developers about the kinds of real-world applications one might build using their technology. A particularly interesting example was that of a zero-knowledge, encrypted cloud calendaring system (which we do not believe currently exists). Crypton provides the framework for building such a system, and we are excited to see what other previously-infeasible applications creative developers will build upon the features Crypton provides.

Observations

Our observations are broken out below. They are based on different high-level attack perspectives and intended targets and then separately for our investigation of Crypton's SRP implementation.

Server-to-Client Attacks

A key component of the Zero-Knowledge threat model is the server as a malicious actor. To simulate untrustworthy server behavior, we modified replies to the client in an attempt to cause the client to do something unsafe. This included altering fields that were not digitally signed and replaying valid requests from previous sessions. We also evaluated the client's parsing of server response for HTML and Javascript code injection vulnerabilities and other common client-side weaknesses. However, due to the smart design of the Crypton client, including safe handling of all server-supplied data, such attacks do not exist.

1	Record Replacement Attack	
	Id	72635
	Type	Implementation
	Risk	Critical
	Impact/Skill Level	Critical/Simple
	Reference	n/a
	Location	/client/src/container.js, Container.prototype.decryptRecord
	Observation	Due to finding 72634, Alice does not verify her public signing key when she receives it from the server. If the server uses this attack, the server can tamper any record that Alice expects to be signed by herself. Further, because the server has Alice's public encryption key, it can create messages for her to decrypt. If the server gives Alice a crafted public signing key upon login, she will use that key to verify any records she expects to be from herself. As a result, the server can tamper with the encrypted message (payloadCiphertext) of any record when it's retrieved. The server could then use this to create new records encrypted with Alice's public encryption key that she can then decrypt. The result is full message replacement -- while the server cannot read the old messages, it can create new messages in their place. Combined with finding 72637, the server could completely control the application Alice is trying to load, as it would be able to read container names Alice is requesting and replace them with different containers that it has crafted.
	Recommendation	Remediate finding 72634.

2	Man-in-the-Middle Attack	
	Id	72628
	Type	Design
	Risk	Critical
	Impact/Skill Level	Critical/Simple
	Reference	n/a

2 Man-in-the-Middle Attack

Location	/examples/chat3
Observation	[1/30/2014, Vulnerable -> Updated]: This attack builds on findings 72636 and is not a full finding in itself. A malicious server can lie about peer information in order to decrypt and tamper all messages passed using chat3. Server makes its own client, Mallory. When Alice makes a request for /peer/Bob, instead of returning Bob's public keys it returns Mallory's public keys and account id but Bob's username. All messages will now be sent to Mallory instead of Bob. Mallory can now create a connection to Bob and relay all of Alice's messages if desired.
Recommendation	This protocol is under similar limitations as OTR. As in-band identity verification is impossible, users must do out-of-band fingerprint verification to ensure they are talking to the person they expect and not Mallory.

3 Encrypted data can be modified or corrupted in transit

Id	72637
Type	Design
Risk	High
Impact/Skill Level	High/Simple
Reference	http://cwe.mitre.org/data/definitions/649.html
Location	client/src/account.js, Account.prototype.unravel()
Observation	The value of 'containerNameHmacKeyCiphertext' is not signed. As a result, a malicious server can craft a 'containerNameHmacKey' that the client will blindly use. The server crafts a containNameHmacKey that it knows. It then encrypts this value with a new symmetric key that it crafts and then encrypts that symmetric key using the public key of the user. The server then returns the forged symKeyCiphertext and the forged containerNameHmacKeyCiphertext to the user upon login. The server can now decrypt all Hmac-encrypted containerName requests, allowing it to control what containers are returned to the user.
Recommendation	Sign all ciphertext blobs and verify signatures before decryption.

4 Denial of Service

Id	72634
Type	Implementation
Risk	Medium
Impact/Skill Level	Medium/Simple
Reference	http://cwe.mitre.org/data/definitions/730.html
Location	/client/src/account.js
Observation	The code in Account.prototype.unravel() does not verify that the ECC public signing key sent by the server is the correct pair for the ECC private signing

4	Denial of Service	
		key that it decrypts. A malicious server can trick a client into using an incorrect public key, causing havoc with incorrectly-signed messages down the road. 1. Client creates an account. 2. Client logs into diary and creates a diary entry. 3. Client logs out. 4. Client logs back into diary, server sends an incorrect but valid signKeyPub point vector. 5. Client views diary but diary entry fails to decrypt. Error in Javascript console: "uncaught exception: CORRUPT: signature didn't check out".
	Recommendation	In the unravel function derive the public key from the private key or at least use the private key to verify that the public key is correct.

5	Denial of Service	
	Id	72633
	Type	Implementation
	Risk	Medium
	Impact/Skill Level	Medium/Simple
	Reference	http://cwe.mitre.org/data/definitions/730.html
	Location	/client/src/account.js
	Observation	The code in Account.prototype.unravel() does not verify that the ECC public key sent by the server is the correct pair for the ECC private key that it decrypts. A malicious server can trick a client into using an incorrect public key, causing havoc with incorrectly-signed messages down the road. 1. Client creates an account. 2. Client logs into diary and creates a diary entry. 3. Client logs out. 4. Client logs back into diary, server sends an incorrect but valid pubKey point vector. 5. Client views diary but diary entry fails to decrypt. Error in Javascript console: "Error: Cannot verify ciphertext".
	Recommendation	In the unravel function derive the public key from the private key or at least use the private key to verify that the public key is correct.

6	Denial of Service	
	Id	72632
	Type	Design
	Risk	Low
	Impact/Skill Level	Low/Simple
	Reference	http://cwe.mitre.org/data/definitions/730.html
	Location	/peer/user

6	Denial of Service	
	Observation	The server can modify the response to a /peer/user request, and in doing so prevent two users from ever communicating with each other. Alice wants to chat with Bob so requests /peer/Bob. The server modifies the response giving a different accountId, different pubKey values, or both. Alice receives this information and attempts to start a conversation with Bob. Message from Alice to Bob either go to the wrong accountId or are encrypted to the wrong public key. As a result, Bob does not receive any messages from Alice but Alice believes they are being received. Further, if Bob tries to initiate communication with Alice, Alice does not see the messages because they are not being sent to the container Alice expects.
	Recommendation	The server is able to lie about the accountId and pubKey of a user when they are requested by a client. Create a mechanism that makes server tampering of this data obvious and report such tampering to the client.

7	Unhandled Exceptions	
	Id	72629
	Type	Implementation
	Risk	Informational
	Impact/Skill Level	Informational/Advanced
	Reference	http://cwe.mitre.org/data/definitions/209.html
	Location	client/src/account.js
	Observation	In Account.prototype.unravel(), functions are not checked for exceptions before being passed to other functions. On line 71: JSON.stringify() throws an exception when this.keypairCiphertext does not contain valid JSON data sjcl.decrypt() throws an exception when the result of JSON.stringify() is not a valid key objection JSON.parse() throws an exception when sjcl.decrypt() doesn't return JSON data Subsequent lines suffer from similar issues.
	Recommendation	Handle the exceptions

Client-local attacks

Crypton's Zero-Knowledge design lends itself well to privacy-focused application. As a result, we consider the case of a shared computer being used to access a Crypton application. In this scenario, we want to ensure that Crypton does not leave sensitive information on the shared computer that another user might be able to find. We also want to ensure that a client can't accidentally leak sensitive

information over the wire. While a few caching-related findings were found, no serious vulnerabilities impacting user privacy were found.

8 Cacheable HTTPS Response

Id	72630
Type	Configuration
Risk	Low
Impact/Skill Level	Low/Simple
Reference	http://cwe.mitre.org/data/definitions/525.html
Location	Various
Observation	Unless directed otherwise, browsers may store a local cached copy of content received from web servers. Some browsers cache content accessed via HTTPS. If sensitive information in application responses is stored in the local cache, then this may be retrieved by other users who have access to the same computer at a future time. Various paths return cacheable responses, some of which contain sensitive data. Example: /inbox/## - This cached response contains the two participants of a given message ID. /peer/XXXX - This cached response contains the username and ID of an intended communication participant, as well as the username of the local user.
Recommendation	The application should return caching directives instructing browsers not to store local copies of any sensitive data. Often, this can be achieved by configuring the web server to prevent caching for relevant paths within the web root. Alternatively, most web development platforms allow you to control the server's caching directives from within individual scripts. Ideally, the web server should return the following HTTP headers in all responses containing sensitive content: Cache-control: no-store Pragma: no-cache

9 Password field with autocomplete enabled

Id	72631
Type	Implementation
Risk	Low
Impact/Skill Level	Low/Simple
Reference	https://www.owasp.org/index.php/Testing_for_Vulnerable_Remember_Password_and_Pwd_Reset_(OWASP-AT-006)
Location	/examples/chat3
Observation	Most browsers have a facility to remember user credentials that are entered into HTML forms. This function can be configured by the user and also by applications which employ user credentials. If the function is enabled, then credentials entered by the user are stored on their local computer and

9 Password field with autocomplete enabled

	retrieved by the browser on future visits to the same application. The stored credentials can be captured by an attacker who gains access to the computer, either locally or through some remote compromise. Further, methods have existed whereby a malicious web site can retrieve the stored credentials for other applications by exploiting browser vulnerabilities or through application-level cross-domain attacks. Considering the user's passphrase is the only item required to decrypt a user's account details, it should be protected wherever possible. Any forms requesting this information should not allow the browser to save it plaintext.
Recommendation	To prevent browsers from storing credentials entered into HTML forms, you should include the attribute autocomplete="off" within the FORM tag (to protect all form fields) or within the relevant INPUT tags (to protect specific individual fields).

10 Use of Insufficiently Random Values

Id	72647
Type	Implementation
Risk	Low
Impact/Skill Level	High/Advanced
Reference	http://cwe.mitre.org/data/definitions/330.html
Location	/client/src/vendor/sjcl.js
Observation	The SJCL function addEntropy() is never called. Due to Javascript's poor track record with random number generation, it might be the case that SJCL is not producing sufficiently random numbers to provide strong security.
Recommendation	Gather entropy data from user input, then use SJCL's addEntropy function to add this data to SJCL's entropy pool.

11 Path Traversal

Id	72639
Type	Implementation
Risk	Informational
Impact/Skill Level	Informational/Simple
Reference	http://cwe.mitre.org/data/definitions/22.html
Location	account login & peer requests
Observation	The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize special elements that can resolve to a location that is outside of that directory. For account creation, usernames are permitted to contain '/', however when login is attempted the / is interpreted within the URL causing the server to reject the request. Thus, accounts can be created and a user can never log into them. Further, if a username is prefixed with '../' the code will send POST requests to other paths on the domain; so far the only effect is to login

11	Path Traversal	
		as '../peer' and have the peer error message display on the login screen. For peer communication, '/' is considered valid by the form but the server rejects the request. This can be used for path traversal, as a username that starts with '../' will send GET requests to other paths on the domain.
	Recommendation	Do not use user-supplied data when constructing file names or paths. Either create unique filenames programmatically or create an enumeration of pre-determined allowed filenames for use. As a last resort, encode or otherwise sanitize user-supplied filenames to ensure that they do not include characters which have special meaning in this context, such as '.', '\', and '/'.

Client-to-Server Attacks

As with all client-server applications, attacks against the server are a serious concern. We investigated this attack surface by modifying valid client requests to the server and evaluated the results. As a result of proper use of query parameterization, no SQL injection vulnerabilities were found. Other common server-side attacks such as command injection were also not found, again due to proper handling of data around sensitive functionality such as command execution.

12	Path information contained in JSON decoding error messages	
	Id	72638
	Type	Configuration
	Risk	Low
	Impact/Skill Level	Low/Simple
	Reference	http://cwe.mitre.org/data/definitions/209.html
	Location	JSON middleware
	Observation	The software generates an error message that includes sensitive information about its environment, users, or associated data. When an invalid JSON string is sent to the server it fails to decode it and produces a stack trace that includes server-side paths. SyntaxError: Unexpected end of input at Object.parse (native) at /home/ubuntu/crypton/server/node_modules/express/node_modules/connect/lib/middleware/json.js:75:25 at IncomingMessage.onEnd (/home/ubuntu/crypton/server/node_modules/express/node_modules/connect/node_modules/raw-body/index.js:109:7) at IncomingMessage.g (events.js:175:14) at IncomingMessage.EventEmitter.emit (events.js:92:17) at _stream_readable.js:920:16 at process._tickCallback (node.js:415:13)
	Recommendation	Use a standard exception handling mechanism to be sure that your application properly handles all types of processing errors. All error

12	Path information contained in JSON decoding error messages
	messages sent to the user should contain as little detail as necessary to explain what happened.

Client-to-Client Attacks

As one of the test applications for this evaluation was a two-party chat program, we explored the possibility of clients sending malicious payloads to each other. This included things like cross-site scripting and other client-side code injection attacks. Our evaluation found no obvious mechanisms to support malicious chat clients from attacking other chat clients. This is in largely due to proper input and output handling of user-supplied data such as chat messages.

SRP

As part of this evaluation, we reviewed the Secure Remote Password (SRP) protocol. While Crypton's implementation follows the SRP specification as defined in RFC 2945, the specification was written in 2000 and has not been updated to reflect modern password cracking methodology. As a result, the below findings are improvements that Crypton can make to the SRP protocol to harden the protocol against today's password cracking landscape.

13	Password Cracking	
	Id	72645
	Type	Design
	Risk	High
	Impact/Skill Level	Critical/Moderate
	Reference	n/a
	Location	n/a
	Observation	SRP does not attempt to prevent dictionary attacks against V, the SRP verifier. V is defined as: x = HASH(salt || HASH(username || ":" || password)) v = g^x mod N An attacker with access to the verifier, the salt, and the username is able to make an off-line password guess with two SHA256 calculations followed by a 2048-bit modular exponentiation. A malicious server could crack the passwords of its users and then decrypt all the account information permitting it to read and modify all encrypted records. A dedicated attacker that knows only the username can precompute z = HASH(username || ":" || guess) for all password guesses. When the attacker eventually gains access to the database, they now only need to compute HASH(salt || z) and then the modular exponentiation to guess a password. This precomputation would significantly reduce the time necessary to find the passwords for the chosen set of targets.

13 Password Cracking

Recommendation	Stored password-replacements should be protected by a hash function that uses a work-factor such as bcrypt or scrypt. Consider replacing SHA256 with one of these functions to greatly improve the security of password storage. Consider including the salt to the inner-hash function to prevent the aforementioned precomputation attack.

14 Improper Authentication

Id	72640
Type	Implementation
Risk	Medium
Impact/Skill Level	Medium/Simple
Reference	http://cwe.mitre.org/data/definitions/287.html
Location	/client/src/core.js
Observation	From RFC 2945, page 5: "If the server receives a correct response it issues its own proof to the client. The client will compute the expected response using its own K to verify the authenticity of the server. If the client responded correctly the server MUST respond with its hash value." Without this additional step, SRP only provides one-way authentication instead of mutual authentication. The server does not send its proof (M2) to the client, it merely responds with "success=true" and then provides the account details. The client does not check to ensure that the server computed the expected response. This may allow for an adversary to trick the client, such as part of a man-in-the-middle attack.
Recommendation	Have the server send "M2 = H(A | M | K)", and have the client verify it matches before processing the account data from the server.

15 Clear Text Secrets

Id	72641
Type	Design
Risk	Medium
Impact/Skill Level	Critical/Advanced
Reference	http://cwe.mitre.org/data/definitions/311.html
Location	/account
Observation	Upon registration, the client sends srpVerifier and srpSalt to the server. The server blindly stores these values as long as the requested username doesn't already exist. An adversary who is able to passively sniff traffic between client and server can record the srpVerifier value. With this value, the adversary can begin an offline password cracking attack against the user's password per finding 72645. Further, an adversary with this value can impersonate the server in future SRP interactions.

15 Clear Text Secrets

	An adversary who is able to actively intercept traffic between client and server can alter the srpVerifier value before it's recorded by the server. This would allow that adversary to conduct a man-in-the-middle attack.
Recommendation	Ensure the channel between client and server cannot be sniffed, such as by using certificate pinning. Another option is for the server to distribute its public key with the client code distribution and have the client encrypt the srpVerifier with the public key before sending it to the server.

16 Clear Text Secrets

Id	72642
Type	Design
Risk	Low
Impact/Skill Level	High/Advanced
Reference	http://cwe.mitre.org/data/definitions/311.html
Location	/account/<user>/answer
Observation	Upon successful login, the server sends srpVerifier and srpSalt to the client. The client doesn't seem to use these values. An adversary able to passively sniff traffic between client and server can record the srpVerifier value. With this value, the adversary can begin an offline password cracking attack against the user's password per finding 72645. Further, an adversary with this value can impersonate the server in future SRP interactions.
Recommendation	Do not send the srpVerifier in cleartext. As this value is not used by the client, it should not be sent upon login.

18 Unhandled Exception

Id	72644
Type	Implementation
Risk	Informational
Impact/Skill Level	Informational/Advanced
Reference	http://cwe.mitre.org/data/definitions/209.html
Location	/client/src/vendor/srp-client.js
Observation	From RFC 2945: "The client MUST abort authentication if B % N is zero." When the server sends a 512-byte string of 0's as the srpB value, the calculateU function throws an exception. This exception is not caught, resulting in a halt of the front-end service. No error is returned to the client.
Recommendation	Handle the exception and return a useful message back to the user.

19 Unhandled Exception

19 Unhandled Exception

Id	72643
Type	Implementation
Risk	Informational
Impact/Skill Level	Informational/Advanced
Reference	http://cwe.mitre.org/data/definitions/209.html
Location	/server/node_modules/srp/lib/srp.js
Observation	From RFC 2945: "The host MUST abort the authentication attempt if A % N is zero." When the client sends a 512-byte string of 0's as the srpA value the server_getS function throws an exception. This exception is not caught resulting in a crash of the back-end service. Nginx then returns a 502 error to the client.
Recommendation	Handle the exception and return a useful message back to the user.

Appendix A: Vulnerability Classification

Impact

The impact of the vulnerability if exploited. This section has five possible values:

Info – The vulnerability affects a future use case of the technology. There is no immediate threat but will be should the system be configured in an anticipated manner.

Low - Provides an attacker with the ability to gain additional information that could be used to further attack systems or clients. No direct access to the data or resources.

Medium - Possible access to systems or servers. Possibility for reputational damage or access to confidential data. No actual access to data was obtained.

High - Direct access to systems or servers. A high likelihood of reputational damage or a direct impact to the data.

Critical – A high impact vulnerability that could be exploited by a worm.

Skill Level to Exploit

The skill level required by an attacker to exploit the vulnerable condition. This section has three possible values:

Simple - Basic understanding of the technology is all that is required. Tools and attack methodologies are easily obtainable from the Internet.

Moderate – Some moderate knowledge of the technology is required. The attacker may need to entice the victim in order to exploit the condition.

Advanced - The attacker has a near complete understanding of the technology and is well able to write her own exploits. Additional interaction with a victim may be required.

When graphically viewed in the following table the risk associated with a discovered vulnerablitiy can be mitigated based on the potential of the vulnerability being exploited.

Impact Rating (Weight)	Advanced (1)	Moderate (2)	Simple (3)
Critical (4)	4	8	12
High (3)	3	6	9
Medium (2)	2	4	6
Low (1)	1	2	3

Skill Level to Exploit Rating (Weight)

Weight

Legend	
Critical	10-12
High	7-9
Medium	4-6
Low	1-3

Appendix B: Project Team

The project team consisted of the following individuals:

Contact	Role	Contact Information
Leviathan		
Mark Stribling	Project Manager	Mark.stribling <at> leviathansecurity.com
Paul Brodeur	Lead Security Consultant	Paul.brodeur <at> leviathansecurity.com
SpiderOak		
David Dahl	Developer	
Alan Fairless	CTO	
Ethan Rishon Oberman	CEO	
Cam Pedersen	Software Engineer	