

# Security Assessment the EasyCrypt Encryption Service for the Open Technology Fund

---



## TABLE OF CONTENTS

EXECUTIVE SUMMARY .....	4
Scope and Methodology .....	4
Assessment Objectives.....	4
Findings Overview .....	4
Next Steps .....	4
ASSESSMENT RESULTS .....	5
CRITICAL-RISK FINDINGS .....	6
C1: Persistent Cross-Site Scripting (XSS) .....	6
C2: Insecure CORS Origin Match.....	7
HIGH-RISK FINDINGS .....	10
H1: Redis Key Query Injection.....	10
MEDIUM-RISK FINDINGS.....	13
M1: Temporary 502 Denial of Service Condition Identified .....	13
M2: Authentication System Does Not Protect Against Brute-Force Attacks.....	14
M3: Sensitive Information Stored in LocalStorage .....	15
LOW-RISK FINDINGS.....	17
L1: Application Functionality Can Be Used to Confirm Valid Accounts .....	17
L2: Cryptographic Secrets Stored in Source Code.....	18
L3: Insecure Mailgun WebHook Implementation .....	19
L4: Weak User Password Requirements .....	20
L5: Credentials for External Services Stored in Source Code .....	20
L6: Content Security Policy Not Implemented .....	21
L7: HTTP Strict Transport Security Not Implemented.....	22
L8: Missing X-XSS-Protection Header.....	23
L9: Missing X-Frame-Options Header .....	23
L10: AES CFB Mode Encryption Used Without Integrity Checking .....	24
INFORMATIONAL FINDINGS.....	26

I1: Use of RoundCube Version with Known Vulnerabilities .....	26
I2: Input Validation Not Used Consistently Throughout Application.....	26
I3: Missing X-Content-Type-Options Header .....	27
APPENDICES .....	29
A1: EasyCrypt Service Description .....	29
A2: Architecture Review .....	29
A3: Architecture Risks .....	30
A4: Test Coverage .....	32

## EXECUTIVE SUMMARY

### Scope and Methodology

IncludeSec performed a security assessment of the EasyCrypt Encryption Service for the Open Technology Fund. The assessment team performed a 3 day effort spanning from May 12th – May 14th, 2017, using a Light Grey Box Assessment Methodology which included a detailed review of all the components described above in a manner consistent with the original Statement of Work (SOW).

### Assessment Objectives

The objective of this assessment was to identify and confirm potential security vulnerabilities within targets in-scope of the SOW. The team assigned a qualitative risk ranking to each finding. IncludeSec also provided remediation steps which Open Technology Fund could implement to secure its applications and systems.

### Findings Overview

IncludeSec identified 19 categories of findings. There were 2 deemed a “Critical-Risk,” 1 deemed a “High-Risk,” 3 deemed a “Medium-Risk,” and 10 deemed a “Low-Risk,” which pose some tangible security risk. Additionally, 3 “Informational” level findings were identified that do not immediately pose a security risk.

IncludeSec encourages Open Technology Fund to redefine the stated risk categorizations internally in a manner that incorporates internal knowledge regarding business model, customer risk, and mitigation environmental factors.

### Next Steps

IncludeSec advises Open Technology Fund to remediate as many findings as possible in a prioritized manner and make systemic changes to the Software Development Life Cycle (SDLC) to prevent further vulnerabilities from being introduced into future release cycles. This report can be used by Open Technology Fund as a basis for any SDLC changes. IncludeSec welcomes the opportunity to assist Open Technology Fund in improving their SDLC in future engagements by providing security assessments of additional products.

## ASSESSMENT RESULTS

At the conclusion of the assessment, Include Security categorized findings into four levels of perceived security risk: critical, high, medium, or low. Any informational findings for which the assessment team perceived no direct security risk, were also reported in the spirit of full disclosure. The risk categorizations below are guidelines that reflect best practices in the industry and may differ from Open Technology Fund's internal perceived risk. It is common and encouraged that all clients recategorize findings based on their internal business risk tolerances. All findings are described in detail within the final report provided to Open Technology Fund.

**Critical-Risk** findings are those that pose an immediate and serious threat to the company's infrastructure and customers. This includes loss of system, access, or application control, compromise of administrative accounts or restriction of system functions, or the exposure of confidential information. These threats should take priority during remediation efforts.

**High-Risk** findings are those that could pose serious threats including loss of system, access, or application control, compromise of administrative accounts or restriction of system functions, or the exposure of confidential information.

**Medium-Risk** findings are those that could potentially be used with other techniques to compromise accounts, data, or performance.

**Low-Risk** findings pose limited exposure to compromise or loss of data, and are typically attributed to configuration issues, and outdated patches or policies.

**Informational** findings pose little to no security exposure to compromise or loss of data which cover defense-in-depth and best-practice changes which we recommend are made to the application.

The findings below are listed by a risk rated short name (e.g., C1, H2, M3, L4, I5) and finding title. Each finding includes: Description (including proof of concept screenshots and lines of code), Recommended Remediation, and References.

## CRITICAL-RISK FINDINGS

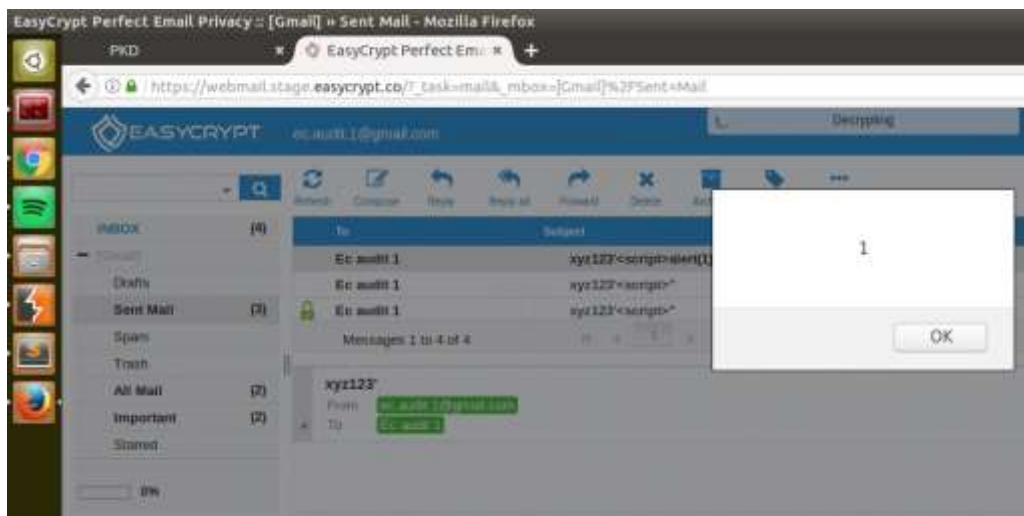
### C1: Persistent Cross-Site Scripting (XSS)

#### **Description:**

Persistent, or stored, cross-site scripting (XSS) occurs when data provided to a web application by a user is first stored persistently within back-end data stores and then displayed to users in a web page without being correctly encoded within the context where it is displayed. An attacker could store malicious client-side executable code to be rendered and executed in a client browser later. Such code might perform actions on behalf of a victim user or compromise their session or account.

There is one instance of persistent XSS discovered in the EasyCrypt application.

For example, malicious code can be sent to the victim through an email with the following subject: **123'<script>alert(1)</script>**. The screenshot below shows a JavaScript alert dialog generated when the victim reads the specially forged email using the EasyCrypt web application:



The JavaScript code is executed when the email is both previewed and read in a different tab. The preview is feature is run when the user clicks on a specific email in the list once, and the email is opened in a different tab when the user double-clicks the email.

An attacker could use this vulnerability to gain full control of the user's EasyCrypt password, which is stored in **localStorage** and accessible through JavaScript. Using this password, the attacker can read all of the user's emails, download the PGP private key and decrypt all emails encrypted using it.

The vulnerability seems to be present in the way the client-side application modifies the DOM in order to display the subject. The root cause was not investigated because it was out of the scope of this engagement.

***Recommended Remediation:***

Cross-Site Scripting vulnerabilities can be reliably prevented with a combination of strict input validation and encoding of all HTML special characters in potentially malicious data. Encoding is generally done directly before a web application or client-side script displays the data, and many programming languages have built-in functions or libraries which provide this encoding (in this context, also called quoting or escaping).

Note that suitable encoding can be applied depending on where the user-supplied data is found. For instance, data appearing inside an HTML block can be HTML-encoded, while data appearing inside a JavaScript block should be JavaScript-encoded.

***References:***

[OWASP Cross-Site Scripting \(XSS\) Page](#)  
[XSS \(Cross-Site Scripting\) Prevention Cheat Sheet](#)  
[Development and Implementation of Secure Web Applications](#)

## **C2: Insecure CORS Origin Match**

***Description:***

Cross-Origin Resource Sharing ([CORS](#)) allows JavaScript code running in the context of domain A to perform HTTP requests to domain B and get access to the HTTP response body. This communication is only possible if the resource owner (domain B) follows the CORS protocol and specifically allows the consumer domain (domain A) in the **Access-Control-Allow-Origin** HTTP response header.

Another feature from the CORS specification allows the JavaScript code running in the context of domain A to include cookies for the domain B in the HTTP requests, this allows the client to consume the services offered by the resource owner using any sessions available in the browser.

A misconfiguration was identified in the way that EasyCrypt implements the Origin header match, which allows an attacker to bypass any restrictions and gain access to sensitive information such as user emails. First, let's analyze an HTTP request sent by the browser:

```
GET /?_task=mail&_caps=... HTTP/1.1
Host: webmail.stage.easycrypt.co
Origin: https://auth.stage.easycrypt.co
```

The **Origin** header is sent by the browser in the request and represents the consumer domain (Domain A in the previous paragraphs). The **Host** header is the target of the HTTP request and represents the resource owner (B). In this case, both domains are under the control of EasyCrypt and access is granted by adding two headers to the response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://auth.stage.easycrypt.co
Access-Control-Allow-Credentials: true
```

An issue was identified in the way the value of the **Origin** header is matched in the EasyCrypt application, which allows the following interaction:

```
GET /?_task=mail&_caps=... HTTP/1.1
Host: webmail.stage.easycrypt.co
Origin: "https://webmailxstagexeasycrypt.co":https://webmailxstagexeasycrypt.co

HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://webmailxstagexeasycrypt.co
Access-Control-Allow-Credentials: true
```

This vulnerability can be exploited by an attacker by registering the **webmailxstagexeasycrypt.co** domain, creating custom JavaScript code that will use CORS to read emails from **webmail.stage.easycrypt.co** and then convince EasyCrypt users into clicking on a link to the attacker's domain. If the victim has an active EasyCrypt session then the attacker will be able to access all his emails.

The issue also affects **pkd.stage.easycrypt.co**, in this case the severity is lower, but can still be exploited to extract user information such as encrypted versions of the PGP private key.

Most likely this problem occurs because of an insecure usage of regular expressions to match the allowed domains. The application code is most likely using a regular expression such as **.\*.easycrypt.co** when it should be using **.\*\.easycrypt.co**. The assessment team recommends conducting an in-depth penetration test that involves a review of the source code to uncover other issues such as this.

#### ***Recommended Remediation:***

The assessment team recommends using string comparisons to match the **Origin** header received in HTTP requests with a list of allowed domains.



Finally, create a list of all the resources which can be consumed using CORS requests and analyze if there is a functional requirement associated with it. In the **webmail.stage.easycrypt.co** application it might be possible to simply disable all CORS features.

***References:***

[Exploiting CORS Misconfigurations](#)

[CWE-942: Overly Permissive Cross-domain Whitelist](#)

## HIGH-RISK FINDINGS

### H1: Redis Key Query Injection

#### *Description:*

Two instances of Redis key query injection were identified in the EasyCrypt application. This vulnerability occurs when user-controlled variables are used to create a string which is then used to query Redis keys.

#### **Instance #1**

The `ec_jwt_session` class defined in `easycrypt-ec-webmail-0a28dd7be148/plugins/ec_private/jwt/ec_jwt.php` creates the `session_key` attribute using user-controlled variables:

```
$this->session_key = "session-".$this->ec_jwt->get_secret()."-".$this->ec_jwt->get_email_user();
```

Which is then used to perform Redis queries:

```
if ($data = $this->redis->hGetAll($this->session_key)) {
```

This vulnerability was combined with the hard-coded encryption and signing secret for JWT and exploited in order to retrieve PGP keys for all users from the stage environment:

```
User with sub 1350122 does not exist
User with sub 1350123 does not exist
Got key with fingerprint 4BD57CCE62532CE41A051690A19385A5218D97CF for pic...@yandex.ru
User with sub 1350125 does not exist
...
User with sub 1350138 does not exist
Got key with fingerprint 5F011049C98405F8255858DD0299B26D5AEDD680 for rud...@gmail.com
Got key with fingerprint 0AF559D756C9F4DE1F0738D1B53B2C2224E44AE6 for chr...@gmail.com
```

#### **Instance #2**

This instance was identified in the `/signup` resource exposed by the `account` micro-service. It can be confirmed by sending these requests:



The screenshot shows the EASYCRYPT security setup interface. At the top, the EASYCRYPT logo is displayed, followed by the text "Welcome, kmzn@aol.jp!". Below this, a progress bar indicates "STEP 1" is active, with the label "Security setup". The main content area contains two password input fields: "Choose password" and "Confirm password". The "Choose password" field has a tooltip that reads "(at least 8 characters, minimum 2 digits and 2 letters)". Below the input fields, a note states "Currently only Chrome, Firefox, Safari (non-Private mode) and Tor browsers are supported". A green "Next" button is positioned to the right of the "Confirm password" field. At the bottom, a progress bar indicates "STEP 2" is the next step, labeled "Connect to your email service".

Figure 1: [https://account.stage.easycrypt.co/signup?id=\\*](https://account.stage.easycrypt.co/signup?id=*)



This screenshot is identical in layout to Figure 1, showing the EASYCRYPT security setup interface. The user's email address is "b.a.schram@hotmail.com!". The "Choose password" field has a tooltip that reads "(at least 8 characters, minimum 2 digits and 2 letters)". The note below the fields states "Currently only Chrome, Firefox, Safari (non-Private mode) and Tor browsers are supported". A green "Next" button is located to the right of the "Confirm password" field. The progress bar at the bottom shows "STEP 2" as the next step, labeled "Connect to your email service".

Figure 2: [https://account.stage.easycrypt.co/signup?id=a\\*](https://account.stage.easycrypt.co/signup?id=a*)

This vulnerability instance can be exploited to identify application users and add new PGP keys to their accounts.

### Note

More instances of this vulnerability are very likely to be present in the application. The security assessment team recommends a full penetration test with source code review be conducted to identify and fix all such issues.

***Recommended Remediation:***

The assessment team recommends implementing strict input validation for any user-controlled parameter which will be used in Redis queries. Input should be validated using a whitelist regular expression which only allows **a-zA-Z0-9**. Another potential solution is to concatenate all the user-controlled parameters into a string, apply the SHA1 hashing algorithm, and use the result as the search key for Redis queries.

***References:***

[Injection Flaws](#)

## MEDIUM-RISK FINDINGS

### M1: Temporary 502 Denial of Service Condition Identified

**Description:**

A Denial of Service vulnerability was identified during the security assessment of the EasyCrypt application. The issue was triggered using Burp Suite's automated vulnerability scanner and resulted in the **account.easycrypt.co** domain returning HTTP code 502 for all requests:

```
HTTP/1.1 502 Bad Gateway
Server: nginx/1.12.0
Date: Fri, 12 May 2017 19:45:29 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 173
Connection: close
Access-Control-Allow-Origin: https://account.easycrypt.co
Access-Control-Allow-Credentials: true

<html>
<head><title>502 Bad Gateway</title></head>
<body bgcolor="white">
<center><h1>502 Bad Gateway</h1></center>
<hr><center>nginx/1.12.0</center>
</body>
</html>
```

The servers were inaccessible for at least 15 minutes after stopping the automated scan process.

The root cause of this issue couldn't be identified due to the short period for this security review, but given the low bandwidth required to trigger this denial of service, it seems to be a performance issue in the application code.

An attacker with almost no technical skills could exploit this issue to stop users from accessing the application.

**Recommended Remediation:**

The assessment team recommends identifying the root cause which made the service unavailable and fixing it.

Other recommendations such as improving monitoring and alerting for all micro-services, auto-scaling when servers have a high load, and detecting unavailable instances will lower the impact of this type of issue in the future.

## References:

[CWE-400: Uncontrolled Resource Consumption \(Resource Exhaustion\)](#)

## M2: Authentication System Does Not Protect Against Brute-Force Attacks

### Description:

The authentication system does not prevent brute-force attacks against accounts. Given a large number of authentication attempts, an attacker may be able to use an automated brute-force attack to successfully guess users' authentication credentials.

A simple proof-of-concept script was developed to perform a brute-force attack on the application. The script performs 200 invalid login attempts and then sends the valid password:

```
[include:/easycrypt/tools] 5s $ python bruteforce.py
Invalid password: 0
Invalid password: 1
Invalid password: 2
...
Invalid password: 197
Invalid password: 198
Invalid password: 199
User password found! vX...lJ
```

The vulnerability exists in the following code from **easycrypt-ec\_auth/common/gatekeeper.py**:

```
class Authenticator(object):
    @staticmethod
    def login_user_pass(email, password, session):
        user = session.query(User).filter(User.email == email).first()
        if user is None:
            ECLog.log("Unable to find user by email: {0}".format(email), ECLog.WARNING)
            raise ECException(falcon.HTTP_401, ECException.FAILED_STATUS, "Unauthorized")

        # If user not active, do not allow to login
        if user.active == 0:
            ECLog.log("Inactive user try to login: {0}".format(email))
            return False, False

        hashed = str(user.password)
        if bcrypt.hashpw(str(password), hashed) == hashed:
            return True, user

        return False, False
```

### Recommended Remediation:

The assessment team recommends employing anti-automation for the authentication layer. One example might be requiring an authenticating user to complete a CAPTCHA after a small number of failed authentication attempts. This makes a brute-force authentication attack impossible to automate and greatly reduces the risk.

Another potential solution to this issue is to require all users to use multi-factor authentication for their accounts.

### References:

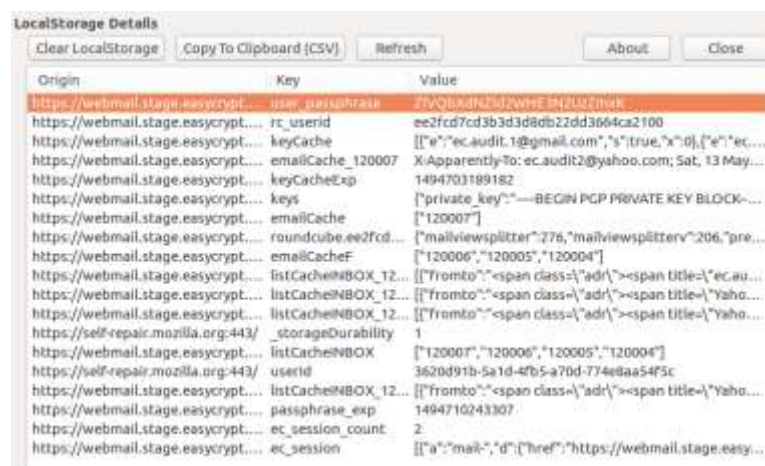
- [OWASP Authentication Cheat Sheet](#)
- [Blocking Brute Force Attacks](#)
- [Development and Implementation of Secure Web Applications](#)

## M3: Sensitive Information Stored in LocalStorage

### Description:

LocalStorage is a persistence feature added to the browsers as part of HTML5. LocalStorage provides a key-value store which is persisted on disk and only removed when the application developer specifically requests it.

Sensitive information, including the user's EasyCrypt password, emails, and PGP keys are stored in the browser's localStorage without encryption. The EasyCrypt user password is the most critical piece of information since it can be used to login into the EasyCrypt account and decrypt any existing emails.



Origin	Key	Value
https://webmail.stage.easycrypt...	user_passphrase	ZVQ0b3dH2dd2WebE3MUM7mk...
https://webmail.stage.easycrypt...	rc_userid	ee2fcd7cd3b3d3d8db22dd3664ca2100
https://webmail.stage.easycrypt...	keyCache	[["e":"ec.audit.1@gmail.com", "s":true, "x":0], ["e":"ec...
https://webmail.stage.easycrypt...	emailCache_120007	XApparently-To: ec.audit2@yahoo.com; Sat, 13 May...
https://webmail.stage.easycrypt...	keyCacheExp	1494703189182
https://webmail.stage.easycrypt...	keys	[["private_key": "-----BEGIN PGP PRIVATE KEY BLOCK-----
https://webmail.stage.easycrypt...	emailCache	[["120007"]
https://webmail.stage.easycrypt...	roundcube.ee2fcd...	[["mailviewsplitter":276, "mailviewsplitter":206, "pre...
https://webmail.stage.easycrypt...	emailCacheF	[["120006", "120005", "120004"]
https://webmail.stage.easycrypt...	listCache@NBOX_12...	[["Fromto": "<span class='ad'><span title='<ec au...
https://webmail.stage.easycrypt...	listCache@NBOX_12...	[["Fromto": "<span class='ad'><span title='<Yaho...
https://webmail.stage.easycrypt...	listCache@NBOX_12...	[["Fromto": "<span class='ad'><span title='<Yaho...
https://self-repair.mozilla.org:443/	_storageDurability	1
https://webmail.stage.easycrypt...	listCache@NBOX	[["120007", "120006", "120005", "120004"]
https://self-repair.mozilla.org:443/	userid	3620d91b-5a1d-4fb5-a70d-774e8aa54f5c
https://webmail.stage.easycrypt...	listCache@NBOX_12...	[["Fromto": "<span class='ad'><span title='<Yaho...
https://webmail.stage.easycrypt...	passphrase_exp	1494710243307
https://webmail.stage.easycrypt...	ec_session_count	2
https://webmail.stage.easycrypt...	ec_session	[["a": "mail-", "d": ["href": "https://webmail.stage.easy...

An attacker with access to the user's computer will be able to read the sensitive information from the disk.

***Recommended Remediation:***

The assessment team recommends using **sessionStorage** to store application secrets. Information stored in **sessionStorage** will be removed once the browser tab is closed, reducing the attack window for attacks on the user's computer.

***References:***

[HTML5 Security Cheat Sheet](#)



## LOW-RISK FINDINGS

### L1: Application Functionality Can Be Used to Confirm Valid Accounts

#### **Description:**

The EasyCrypt application's login functionality is implemented in such a way that would allow an anonymous user to confirm system user's email addresses as valid accounts. In the current implementation, the application responds differently depending on whether the input supplied was recognized as associated with a valid user or not. This behavior could be used as part of a two-stage automated attack. During the first stage, an attacker would iterate through a list of account names to determine which correspond with valid accounts. During the second stage, the attacker would use a list of common passwords to attempt to brute force credentials for accounts that were recognized by the system in the first stage.

#### **Proof of Concept**

The EasyCrypt application presented the message below when the supplied input was recognized as associated with a valid user account:

```
POST /auth HTTP/1.1
Host: auth.stage.easycrypt.co

{"email":"ec.audit2@yahoo.com","password":"846586ce64193ce8fdd50a4b5bc65ffc8b1a9c1011b08dba4f8ffbe2f67e3ec8"}

HTTP/1.1 401 Unauthorized

{
  "status": "failure",
  "debug": "Authentication with password failed: ec.audit2@yahoo.com",
  "message": "Unauthorized"
}
```

And presented the message below when the input was not recognized by the system:

```
POST /auth HTTP/1.1
Host: auth.stage.easycrypt.co

{"email":"foo@yahoo.co","password":"846586ce64193ce8fdd50a4b5bc65ffc8b1a9c1011b08dba4f8ffbe2f67e3ec8"}

HTTP/1.1 401 Unauthorized

{
  "status": "failure",
  "message": "Unauthorized"
}
```

**Recommended Remediation:**

The assessment team recommends adjusting the functionality described above to ensure that it does not disclose whether an account exists or not. This can be accomplished by presenting a consistent error message in both cases.

**References:**

[Username Enumeration Vulnerabilities](#)  
[Testing for User Enumeration and Guessable User Account](#)  
[Development and Implementation of Secure Web Applications](#)

## L2: Cryptographic Secrets Stored in Source Code

**Description:**

Cryptographic secrets used for signing and encrypting JWT in the staging environment were found within the EasyCrypt application's source code. As access to the source code may be exposed due to another exploit (e.g., file traversal) or via a shared source code repository, any attacker or malicious insider would have access to these secrets. This may enable them to attack the cryptographic systems used by the application.

The hard-coded secrets were identified in **easycrypt-ec-webmail/plugins/ec\_private/config.inc.php** and are only used in the staging environment.

```
// Shared AES key
$config['shared_key'] = '934...f01';

// JWT Secret
$config['jwt_secret'] = '...';
```

Since the staging environment is publicly accessible an attacker could use this information to perform more advanced attacks on the staging environment and then potentially pivot into the production site.

The encrypted and signed JWT store important information such as the user's ID and role. Attackers which can forge valid JWTs will, for some specific application features, be able to impersonate an application user or elevate their role to administrator.

**Recommended Remediation:**

The assessment team recommends reviewing the design of the cryptographic system and considering other implementations that would not involve storing secrets in the source code. For example, the EasyCrypt application could extract secrets to a separate file outside the source code tree. It could then protect that file with suitable file system level access controls and an encryption mechanism that requires user interaction to decrypt it, such as a key store that requires a user-provided password upon system restart.

**References:**

[Cryptography API: Next Generation](#)

[The Cryptography API, or How to Keep a Secret](#)

[Windows Data Protection](#)

[Keychain Services Tasks for Mac OS X](#)

[Essentials of the Java Programming Language: A Hands-On Guide, Part 2, Lesson 3:](#)

[Cryptography](#)

[Cryptography with Java, Cryptographic Keys](#)

### L3: Insecure Mailgun WebHook Implementation

**Description:**

EasyCrypt uses the Mailgun service to handle emails sent to registerpublickey@easycrypt.co. When an email is received by Mailgun for this email account an HTTP POST request is sent to EasyCrypt servers, this request is then handled by **PublicKeyRegistration.on\_post**.

Using source code review it was possible to identify that the webhook handler does not authenticate the received information in any way before processing it. This allows an attacker to send specially crafted information to the controller, which will process it just as if it were sent by Mailgun.

**Recommended Remediation:**

The assessment team recommends implementing the recommendations regarding “Securing Webhooks” included in the [Mailgun webhook documentation](#)

**References:**

[Mailgun webhook documentation](#)

---

## L4: Weak User Password Requirements

### **Description:**

The EasyCrypt application requires users to choose a password with at least 8 characters, containing a minimum of 2 digits and 2 letters. Given the sensitive information protected by this password: emails and private PGP key, the security assessment team believes that the password strength requirements are weak and must be increased.

Weak user password requirements will give attackers a higher probability of guessing a user's password.

### **Recommended Remediation:**

The assessment team recommends reviewing the unique security and UX requirements of the system and explore if the enforcement of stronger passwords with high entropy can be implemented. A suitable password strength policy should contain the following attributes:

- A minimum password length of at least 14 characters
- Require mixed character sets: alpha, numeric, special, mixed case
- Does not contain the username
- Does not contain commonly used passwords such as **password123password**, **12345678901234**, etc.
- Does not contain common words such as: easycrypt

### **References:**

[Weak Password Requirements](#)  
[Password Strength](#)

## L5: Credentials for External Services Stored in Source Code

### **Description:**

Credentials for external services were found within the EasyCrypt source code. As access to the source code may be exposed due to another exploit (e.g., file traversal) or via a shared source code repository, any attacker or malicious insider would have access to these secrets. This may enable them to attack the services used by the application.

The vulnerability was found in these locations:

File	Line Number(s)	Type
easycrypt-pk_dir/tools/cred_encryptor.py	33	MySQL password
easycrypt-ec_auth/config/app.example.ini	10	MySQL password
easycrypt-ec_auth/config/app.example.ini	13	Mailgun API key

**Recommended Remediation:**

The assessment team recommends reviewing the application design and considering other implementations that would not involve storing secrets in the source code. For example, the EasyCrypt application could extract secrets to a separate file outside the source code tree. It could then protect that file with suitable file system level access controls and an encryption mechanism that requires user interaction to decrypt it, such as a key store that requires a user-provided password upon system restart.

**References:**

[Cryptography API: Next Generation](#)

[The Cryptography API, or How to Keep a Secret](#)

[Windows Data Protection](#)

[Keychain Services Tasks for Mac OS X](#)

[Essentials of the Java Programming Language: A Hands-On Guide, Part 2, Lesson 3:](#)

[Cryptography](#)

[Cryptography with Java, Cryptographic Keys](#)

## L6: Content Security Policy Not Implemented

**Description:**

None of the EasyCrypt applications implement a content security policy.

Content Security Policy is a W3C specification which offers the possibility for an application to instruct the client browser about which locations and/or which type of resources are allowed to be loaded within the application. To define a loading behavior, the CSP specification uses “directives” which define a loading behavior for a target resource type.

CSP is a very effective way to prevent Cross-Site Scripting vulnerabilities and also forces developers to split view and controller code.

**Recommended Remediation:**

1. Understand the CSP benefits, complexities and different types of directives. Special attention should be paid to the latest CSP implementation mode which uses nonces, as it is easier to implement and very effective.
2. Analyze the effort associated with enabling CSP for the different EasyCrypt applications
3. Enable the CSP in **Report-Only mode**. Analyze and fix all the policy errors received by browsers
4. Change the CSP to enforce mode

**References:**

[OWASP – Content Security Policy](#)

**L7: HTTP Strict Transport Security Not Implemented****Description:**

HTTP Strict Transport Security (HSTS) is a web security policy mechanism which helps to protect websites against protocol downgrade attacks and cookie hijacking. It allows web servers to declare that web browsers (or other complying user agents) should only interact with it using secure HTTPS connections, and never via the insecure HTTP protocol. The HSTS Policy is communicated by the server to the user agent via an HTTP response header field named **Strict-Transport-Security**.

None of the following domains implement HSTS:

- **auth.stage.easycrypt.co**
- **pkd.stage.easycrypt.co**
- **webmail.stage.easycrypt.co**
- **account.stage.easycrypt.co**

This situation could be exploited by an attacker to perform protocol downgrade attacks which could be used to intercept and/or modify HTTP traffic between the user and EasyCrypt servers.

**Recommended Remediation:**

The assessment team recommends implementing HSTS in order to protect against Man-in-the-Middle attacks by including the following HTTP response header in all responses sent by the toosheh.org application:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

Special care should be taken before enabling HSTS since an incorrect implementation might lead to a self-inflicted denial of service which will block users from accessing parts of the application. Before enabling HSTS, make sure all the application content, such as CSS, JS and images is accessible over HTTPS and is hosted on the toosheh.org site.

Once the HSTS implementation is complete, consider submitting the domain to Chrome's preload list, which will add extra security for Chrome users since the initial HTTP request is never sent if the site is in the preload list.

**References:**

[HTTP Strict Transport Security – OWASP](#)  
[HTTP Strict Transport Security – Wikipedia](#)

## L8: Missing X-XSS-Protection Header

**Description:**

The EasyCrypt application doesn't set the HTTP **X-XSS-Protection** header. This header enables the XSS filter within the browser so that the browser will prevent certain types of Cross-Site Scripting attacks.

**Recommended Remediation:**

The assessment team recommends adding the missing HTTP response header to all responses, as follows:

```
X-XSS-Protection: 1; mode=block
```

**References:**

[OWASP Secure Headers Project](#)

## L9: Missing X-Frame-Options Header

**Description:**

EasyCrypt's web application is vulnerable to clickjacking. Clickjacking attacks typically use a combination of stylesheets, iframes, and form elements to convince a targeted user that they

---

are interacting with an innocuous page when instead, they are typing into or clicking on an invisible frame controlled by an attacker. A successful clickjacking attack could circumvent cross-site request forgery (CSRF) protections that attempt to confirm transactions with the user, resulting in an unwanted transaction.

***Recommended Remediation:***

Given that the previously explained mitigating risk factors might change in the future, the security assessment team recommends that all HTTP responses from the web server contain these headers:

```
X-Frame-Options: DENY
Content-Security-Policy: frame-ancestors 'none'.
```

These headers indicate to browsers that the response is not allowed to be contained inside an iframe. If a portion of the user interface needs to be framed by another area of the application, then the assessment team recommends using the following header for those areas instead:

```
X-Frame-Options: SAMEORIGIN
Content-Security-Policy: frame-ancestors 'none'.
```

***References:***

- [OWASP Clickjacking Page](#)
- [Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites](#)
- [The X-Frame-Options Response Header](#)
- [Combating ClickJacking With X-Frame-Options](#)

## L10: AES CFB Mode Encryption Used Without Integrity Checking

***Description:***

The EasyCrypt application uses AES' Cipher Feedback (CFB) mode of operation to encrypt and decrypt information in the following source code sections:

- **easycrypt-ec\_auth/common/jwt\_helper.py**
- **easycrypt-pk\_dir/common/jwt\_helper.py**
- **easycrypt-ec-webmail/plugins/ec\_private/jwt/ec\_jwt.php**

AES' CFB operation mode is considered more secure than ECB and CBC, but it does not verify the integrity of the data received before decrypting it, which could be abused by an attacker to perform complex cryptographic attacks on the application.



***Recommended Remediation:***

The assessment team recommends using AES with [Galois/Counter Mode](#) (GCM) which provides both confidentiality and integrity for the information being exchanged.

***References:***

[Galois/Counter Mode](#)

## INFORMATIONAL FINDINGS

### I1: Use of RoundCube Version with Known Vulnerabilities

**Description:**

The EasyCrypt software as a service was created using the RoundCube webmail application. It was possible to identify that the RoundCube version in use is 1.2.1 and that the latest available version in the 1.2 branch is 1.2.4.

The [RoundCube project change log](#) indicates that several vulnerabilities were fixed in releases 1.2.2, 1.2.3 and 1.2.4. These vulnerabilities might not be exploitable in EasyCrypt because they affect specific areas of the RoundCube code which are not in use. These assumptions were not confirmed by the security assessment team because they were out of the scope of the engagement.

**Recommended Remediation:**

The assessment team recommends:

- Update to the latest stable RoundCube version
- Actively monitor new RoundCube releases and update to the latest release if new vulnerabilities are identified
- Actively monitor any security related issues or pull requests in the RoundCube GitHub repository

**References:**

[Top 10 2013 / A9 / Using Components with Known Vulnerabilities](#)

### I2: Input Validation Not Used Consistently Throughout Application

**Description:**

The input validation performed by the various REST APIs and web applications which compose the EasyCrypt application is weak, situation which lead to various application security vulnerabilities and will lead to more issues in the future.

For example, the **UsersKeyResource.on\_post** controller method uses the **key\_importer\_resource\_on\_post\_on\_patch** function to validate the email parameter, but does nothing to validate the **public\_key** and **private\_key** attributes:

---

```
class UsersKeyResource(BaseResource):
    @falcon.before(ec_validators.key_importer_resource_on_post_on_patch)
    def on_post(self, req, resp):
        private_key = None
        user = req.context['user']()
        public_key = req.context['doc']['public_key']
        if 'private_key' in req.context['doc']:
            private_key = req.context['doc']['private_key']
```

```
def key_importer_resource_on_post_on_patch(req, resp, resource, params):
    validation = {'email': [Required, Pattern("^S+@S+\\.S+$")]}
    is_valid, msg = validate(validation, req.context['doc'])
    if is_valid is False:
        raise ECException(falcon.HTTP_400, "error", msg)
```

Also, the regular expression `^S+@S+\\.S+$` accepts email addresses which are not valid according to the RFC.

#### ***Recommended Remediation:***

The assessment team recommends implementing strict input validation using white lists for all inputs accepted by the application. Input validation is the corner-stone of application security and will prevent most vulnerability classes if properly implemented.

#### ***References:***

[Input Validation](#)

### **I3: Missing X-Content-Type-Options Header**

#### ***Description:***

The EasyCrypt application does not set the HTTP header **X-Content-Type-Options** which prevents Internet Explorer and Google Chrome from content-sniffing and interpreting a response which differs from the declared content-type. Setting this header reduces exposure to drive-by download attacks and sites serving user-uploaded content that, by clever naming, could be treated by MSIE as executable or dynamic HTML files.

#### ***Recommended Remediation:***

Configure the nginx server to add the **X-Content-Type-Options** header with value to **nosniff** to all HTTP responses:

```
X-Content-Type-Options: nosniff
```

***References:***

[OWASP – List of useful HTTP headers](#)

## APPENDICES

### A1: EasyCrypt Service Description

EasyCrypt allows users to send and receive PGP-encrypted emails using a software as a service model which integrates with Gmail, Outlook and any email service providing IMAP. This integration allows users to continue using their existing email address and increase the security of their communications by encrypting emails end to end.

One of EasyCrypt's main features is that all the technical details related to encrypting emails and attachments, sharing PGP keys, verifying email signatures and decrypting email contents are hidden from the user and performed using cryptography best practices.

Users consume the EasyCrypt service by logging into their webmail account at <https://webmail.easycrypt.co/login/>, or through the Tor hidden service provided at <http://webmail.ezcrypt2dgcicxqj.onion/login>. The main features provided by the application are:

- Generation of PGP keys from the browser
- Secure storage of public and private PGP key
- Sending and receiving PGP-encrypted emails using existing email address
- Sending and receiving clear-text emails using existing email address
- Import and export PGP keys

Some features, such as end-to-end metadata protection and perfect anonymity are not coded completely and were not tested.

### A2: Architecture Review

The EasyCrypt software as a service is built based on the Open Source RoundCube project, a flexible and extensible webmail application developed in the PHP language. EasyCrypt developers modified the RoundCube source, created new plugins and skins to add security and encryption. The webmail application is available at **webmail.easycrypt.co** and **webmail.ezcrypt2dgcicxqj.onion** and is the main entry point for all users.

The RoundCube version used to build EasyCrypt is 1.2.1.

The software as a service uses a micro-service architecture, these domains are also part of the architecture and provide critical features:

- **auth.easycrypt.co**: User authentication process
- **pkd.easycrypt.co**: PGP key directory
- **account.easycrypt.co**: User account settings and sign-up process

These micro-services expose both REST APIs and small web applications; and are consumed using [CORS](#) requests generated by other domains in the EasyCrypt architecture or by clicking on a link and browsing to a resource which generates HTML content.

The applications use a MySQL database to persist information such as the user's email address, authentication credentials, and PGP keys. Session information is stored in a Redis cache and is removed when the TTL expires.

[OpenPGP.js](#) is used to handle encryption and decryption of emails in the user's browser, and [Crypto.js](#) is used to provide hashing functions used during the login process.

## References

[How it works](#)

[Under the hood](#)

## A3: Architecture Risks

### Important Note

During the security assessment, the security assessment team found that EasyCrypt developers implemented the service features explained in the home page, FAQ and product documentation following the guidelines explained in the [under the hood](#) document. In other words, the analyzed version of EasyCrypt securely stores email provider credentials, EasyCrypt account password, and private PGP keys and implements a secure process for authentication and email encryption.

The risks and attack scenarios explained in the following sections are inherent to EasyCrypt's architecture. There is no indication that these attack scenarios are actively exploited at the moment, but they are technically feasible and users should take them into account before using the EasyCrypt service.

### Email Provider Credentials

When a new user signs up for the EasyCrypt application he needs to provide his email credentials: in the form of an OAuth2 access and refresh token or an IMAP username and password. These credentials are encrypted with a key only known to the user and stored only in EasyCrypt's database.

If an attacker gains access to the information stored in the database the only potential attack is to brute-force the encryption, which is unfeasible and will most likely take tens or hundreds of years.

Since EasyCrypt needs to access the clear-text version of the credentials in order to consume the email providers and retrieve the user's emails, some code sections have access to this very sensitive piece of information. An attacker who gained access to an application server or can introduce changes to the source code repository could potentially modify the application in such a way that clear text credentials are sent to an attacker-controlled server.

The code sections which use the clear-text credentials are:

- OAuth2 handler which receives the access and refresh token during user sign-up and token refresh
- Methods used to read and send emails using a service provider

Please note that in this scenario the attacker would only be able to retrieve clear-text credentials for users which are, at the moment of the attack, using the previously enumerated features. This reduces the attack impact, but still affects any user which has an active session in the EasyCrypt application during the attack.

### **Clear Text Emails**

EasyCrypt handles two types of emails: PGP-encrypted and clear-text. The PGP-encrypted emails are secured by end to end public key cryptography and the message is private even when intercepted by a third party. A privacy concern exists with the clear-text emails which are sent and received using the EasyCrypt application.

Since EasyCrypt uses the subscriber's credentials to read emails from the email service provider and then shows their contents in the EasyCrypt webmail application, it is trivial to understand that there are sections of the EasyCrypt application which handle clear-text emails. An attacker who gained access to an application server or can introduce changes to the source code repository could potentially modify the application in such a way that clear text emails are sent to an attacker-controlled server.

Please note that in this scenario the attacker would only be able to exfiltrate clear-text emails which are read by an active user while the attack is taking place. This reduces the attack impact, but still affects emails read by any user which has an active session in the EasyCrypt application during the attack.

## Proposed Solution

This application's security can be improved if these rules are followed:

- User credentials should never be in clear-text in EasyCrypt servers
- User emails should never touch EasyCrypt servers

These rules can be implemented at least for the Gmail email service provider by using [OAuth's implicit flow](#) and [Google's CORS API](#).

The OAuth implicit flow can be used to get an access token for the Gmail REST API. The difference in this flow is that the access token never reaches EasyCrypt servers and doesn't require any server-side interactions from EasyCrypt to be generated. The access token is handled by the JavaScript code running client-side at the user's browser. Once received the user could encrypt it using his EasyCrypt public key and send it to EasyCrypt for storage.

Google's REST APIs can be consumed using CORS and an access token. When the user authenticates against EasyCrypt an encrypted bundle containing the access token and private PGP key should be sent to the user. The client-side application can then talk directly to Google REST APIs to retrieve emails and (if necessary) decrypt them.

## OPSEC Risk Minimization for Users

The risks associated with EasyCrypt's architecture can be mitigated by signing up for the service using a new email account which *\*must only be used for PGP-encrypted emails\**. By following this recommendation any email contents which might be leaked by a compromise of EasyCrypt servers will be encrypted, and access to the credentials associated with this new email account will only yield encrypted emails which are of very low value for an attacker.

## A4: Test Coverage

The main engagement goal was to review EasyCrypt's architecture and identify issues in the building blocks, interactions between the micro-services, and insecure usage of cryptographic algorithms. This light-weight high-level review should not be considered an in-depth security assessment. The scope of this review included a cursory review of the following domains:

- **auth.stage.easycrypt.co**
- **pkd.stage.easycrypt.co**
- **webmail.stage.easycrypt.co**
- **account.stage.easycrypt.co**



As well as a brief look into the following source code repositories:

- **easyencrypt-ec\_auth (commit: b42acb56ac45)**
- **easyencrypt-ec\_login (commit: aba5a2d02795)**
- **easyencrypt-ec-webmail (commit: 0a28dd7be148)**
- **easyencrypt-pk\_dir (commit: b65cfe0ff044)**

During the engagement, some vulnerabilities such as Persisting Cross-Site Scripting and Redis Key Query Injection were identified through automated scanning, manual testing, and source code review.

It is important to notice that the goal of this engagement was not to identify vulnerabilities in all of the application resources and parameters; thus it is likely that other implementation-related vulnerabilities are present. The application security assessment team recommends performing an in-depth web application penetration test to identify all vulnerabilities.