RSA SecurID
Research & Implementation

SOMETHING AWESOME

YUTIAN (YOLANDA) LI

Table of Contents

1. Abstract

The Something Awesome project is an eight week personal interest project conducted while complementing the university course COMP6841: Extended Security Engineering and Cyber Security at UNSW, on anything related to security. This project report outlines and analyses the security of the RSA SecurID Authentication System as well as its various mechanisms and explores a simplified implementation.

Security is an emerging field within computer science and wider industries with an extremely difficult problem to solve, the ever changing nature of technology and how to defend against attackers. The problem with security is that there are no absolute guarantees, systems can only ever be resistant to attack but can never be unhackable. As security measures evolve so do the attackers that constantly try to compromise those systems. The first step to understanding security is to understand the problem and understand the solutions to the problems that already exist.

This report will dive into the problems of authentication and methods to solve the age old problem caused by the anonymity of the internet, are you who you really say you are? An existing solution to authentication is RSA SecurID. It is a mechanism developed by RSA, The Security Division of EMC for performing two-factor authentication for a user to a network resource, its unique security feature is the use of an independent hardware security token that is not internet enabled. As of 2003, RSA SecurID had over 70% of the two-factor authentication market with 25 million devices produced to date which could be argued to be of industry standard. However, even though its security strength is recognised the mechanism still suffered a system compromise in March 2011. The report will discuss both the mechanism's effectiveness and ineffectiveness in regards to security.

2. Authentication

2.1 The Problem of Authentication

Authentication across the Internet, which is notorious for its anonymity, straddles both the virtual and the real world. Authentication asks the question of how does something, the computer, inside the system prove the existence and identity of something, the person, outside the system?

Broken authentication is listed at number two on the OWASP Top Ten and is continually a factor of security that is either overlooked or naively implemented. Authentication is increasingly more vulnerable to attack due to security breaches of large corporations and people's habits of password reuse. Attackers have access to hundreds of millions of valid username and password combinations, default administrative account lists of organisations with passwords that users never change. Beyond this they can also automate brute forcing passwords and use dictionary attack tools or exploit unexpired session tokens to gain unintentionally granted access. Attackers only need to gain access to one account to do massive damage, either to that person or to an entire corporation with thousands of employees.

Simple problems can be solved with limiting failed login attempts, not using default credentials especially for admin users, implementing weak-password checks, encouraging passwords to comply with NIST 800-63 B guides and logging all authentication failures. More complex authentication problems can be solved using robust server-side session managers and implementing variants of multi-factor authentication such as two factor authentication.

2.2 Two Factor Authentication

Two Factor Authentication (2FA) attempts to solve the authentication problem of confirming a user's claimed identity by using a combination of two different factors:

1. Something they know
2. Something they have
3. Something they are

Even though there are more factors to authenticate against, all three are still essentially secrets that can be stolen. Meaning that just because it's 2FA does not mean it's not vulnerable to the same issues of being brute-forced, guess, leaked, intercepted and ultimately stolen.

We have to note though that nothing is ever absolutely unhackable, the maximum security that can be achieved is to be unhackable within a reasonable time frame. This is what RSA SecurID Authentication attempts to achieve.

3. RSA SecurID System

3.1 Introduction - What and Why?

RSA SecurID is a time-based authentication system for a user to a network resource that addresses the problems of authentication which is relying on a secret that can be stolen.

It uses the principle of two factor authentication:

1. Something you have (RSA SecurID Token)
2. Something you know (Regular login credentials)

The main mechanism consists of a token, either hardware or software, which is assigned to a computer user and generates an authentication code at fixed intervals (usually 6 - 8 digits at 60 second intervals) using a built-in clock and a factory encoded unique random key (seed) which is also loaded into a corresponding RSA SecurID server once activated. It uses a symmetric encryption algorithm based on AES-128 cryptography standard that relies on a shared secret key (seed) and the current time. User will use this authentication code along with their regular login credentials to be authenticated into the system.

Its increased security derives from two main areas. First, the tamper resistant property of the physical hard token as it is not connected to the internet and needs to be physically stolen to be ‘hacked’. Second, the time-based generation of authentication codes by the token making it extremely difficult to steal by interception or brute force in that time period.

Essentially it is as close to taking the secret, smashing it up, lighting it on fire and burying it whilst doing a ritual dance level of security as possible whilst still being useful.

3.2 Soft vs Hard Tokens

There are two options for SecurID tokens, software (soft) and hardware (hard). Although both appear to serve the same functionality, hard tokens are critical in providing SecurID with its security with its tamper resistance whilst soft tokens can essentially be seen as just another password.

Hard Token - *"Something you have"*

Physical tokens are designed to be tamper resistant, which is a critical property. By design, if a physical token is tampered, the internal storage of the token's symmetric secret key (seed) is lost, to prevent a physical attacker from duplicating the token. As such, it ensures from its production that there can only exist one unique hard token at a time that is being used by one unique user because it is a physical object that is factory embedded with the seed and time that can't be extracted.

As SecurID's security is dependent on a symmetric shared secret (seed) which is critical to be kept secret, hard tokens that can only be distributed in person is able to keep the secret safe during transmission as they have to be physically stolen which would be immediately detectable and are also tamper resistant.

One downside is that physical tokens can be difficult and more costly to distribute to users.

Soft Token - *"Something you know"*

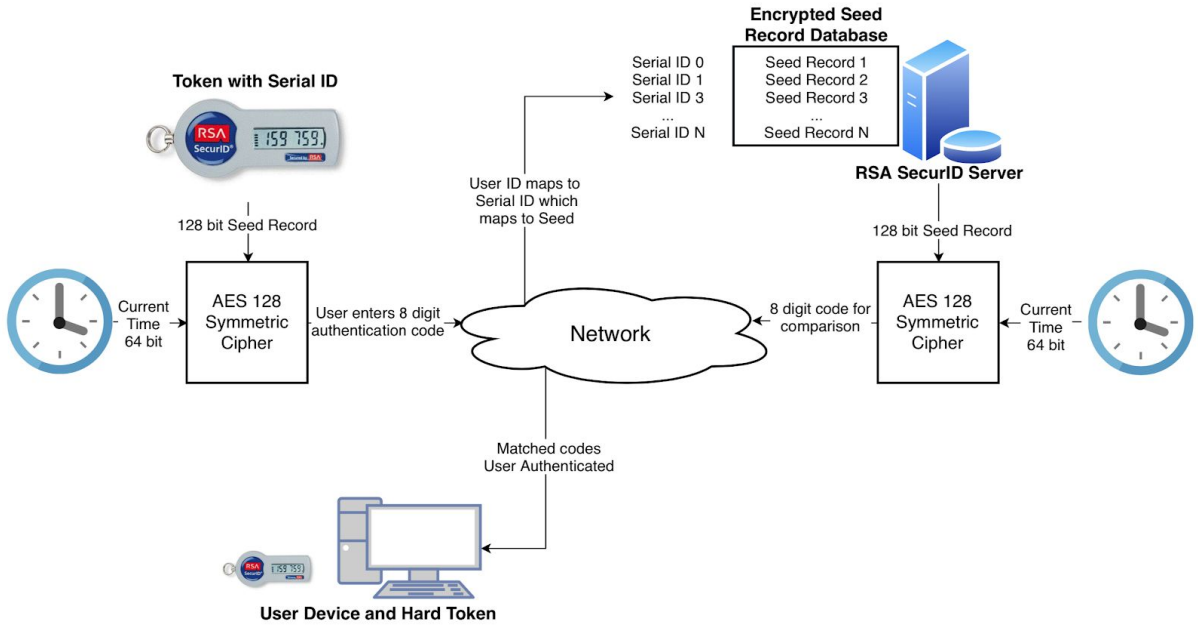
Soft tokens are essentially something you know, and as the basis of RSA SecurID consists of two factors of authentication with something you have and something you know, software tokens in a way can't be counted as an item to have. Software tokens are much more easily spoofed without detection. With soft tokens, the authentication process can prove that a user has the software token, but it cannot prove that only that user has the software token.

Soft tokens are not tamper resistant and are software application code that implements the authentication code generation function and has a seed record that is used to generate the output. Distributing the seed record requires a confidential channel to ensure that it is not perfectly duplicated in transit however most distribution of seed records involve plaintext transmission over the internet and it is commonplace for admins to transmit seed records and initial passphrase side by side. Allowing attackers to duplicate the authentication mechanism and pretend to be the user or lock them out of their account. Even if the secret is safe in transit, a malware ridden device can just as easily allow attacks to extract the seed.

One benefit is that software tokens can be easily distributed over the internet, although it does not offer the safe security as physical tokens do.

3.3 Encryption Algorithm

There are three main parties, RSA SecurID Server, Company Network and User.



The user from a company that needs to be authenticated will be distributed a RSA SecurID hard token which has the Serial ID written on the back and is factory embedded with a symmetric secret key, Seed, and an internal clock. This Serial ID is stored in the company's database and mapped to the user's credentials and unique attributes. This will be used to map the user's User ID which is entered during authentication to the Serial ID which is then used to map to the Seed Record in the RSA SecurID servers to compute token codes for authentication. The internal clock within the token and the RSA SecurID server that does the computation is synched, and if the token is unsynchronised due to internal clock drift, the procedures described in Section 3.4 Time Synchronisation will occur to reestablish synch.

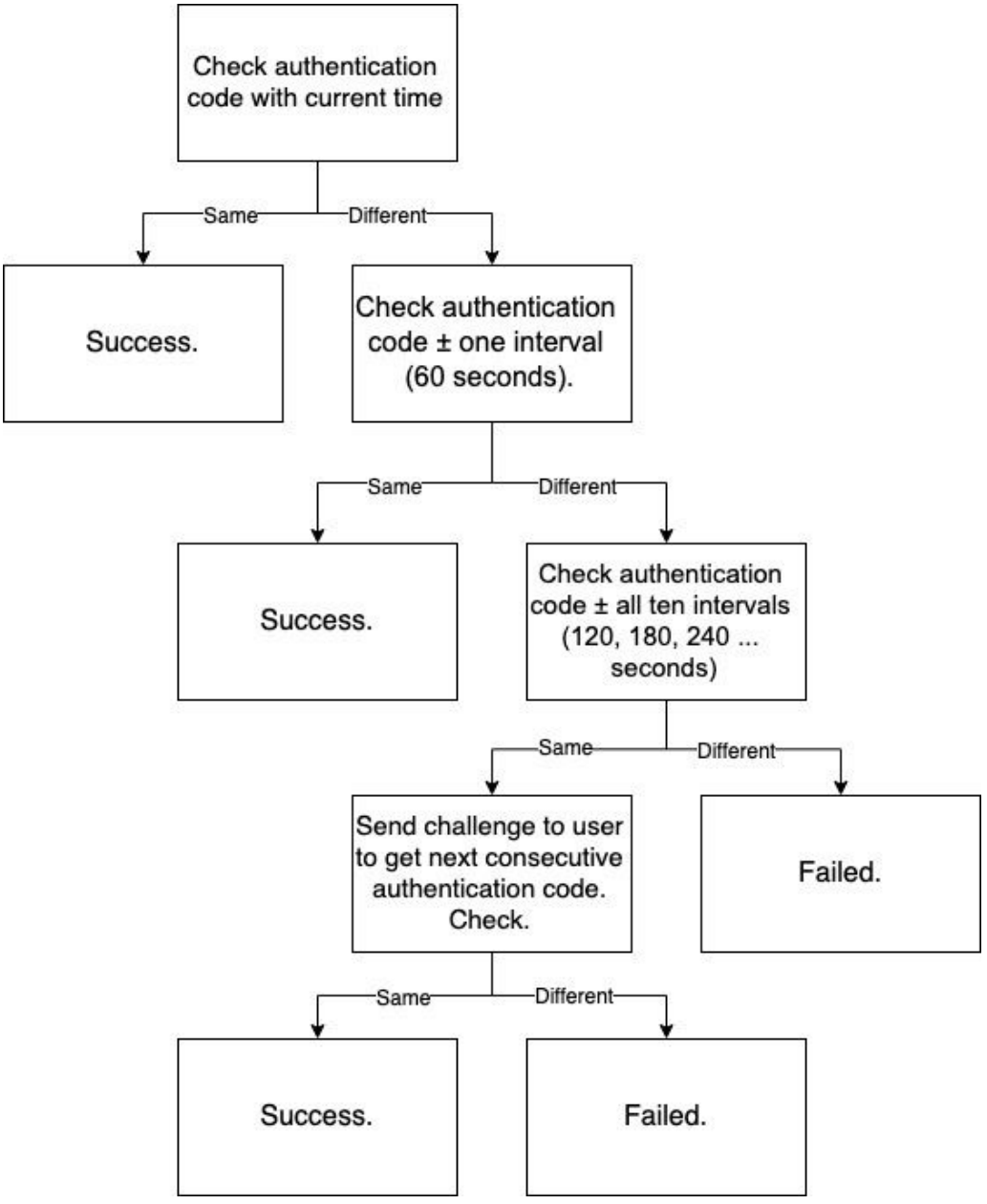
Within the token, for every interval of 60 seconds, a 64-bit timestamp and 128-bit seed record is passed into the SecurID algorithm as input and generates a long length output. From this long length encrypted value, a 6 to 8 digit hash is calculated to be used as the authentication code. The algorithm used in this step is based on the AES-128 symmetric cryptography standard, though it is named RSA SecurID, it does not use the RSA Asymmetric encryption of public and private keys. The authentication code generated is entered into the login page as prompted by the user.

On the server side, the same process is repeated and the generated hash is compared against the one from the user, if both codes match, then the user is authenticated without problem.

3.4 Time Synchronisation

The authentication codes are generated with a combination of the seed and the current clock time of the RSA Server for authenticating. Since hard tokens are tamper resistant and not internet enabled, the built-in clock, no matter how high the accuracy, will gradually drift over time and is unable to self correct to synchronise with the server. To mitigate the issues of an unsynchronised token causing the correct user to fail to be authenticated at all, the system has a time synchronisation functionality.

The server runs through following sequence (hard token system only):



Step 2 Explanation

System has leeway of ± one interval to account for clock inaccuracies over time, system assumes the user token's clock to have drifted and stores these settings for future authentication.

Step 3 Explanation

System attempts to correct any larger clock synchronisation errors with a bigger timing window of ± ten intervals, will test codes from any amount of those intervals until a match is found. If a match is found, it will send a challenge to the user for the next consecutive authentication code from their token.

This ensures if an attacker guessed an authentication code within this time interval once (much more plausible given they have more time to compute), unless they know the very next number in one interval (60 seconds), the server won't authenticate. Since 8-digit passcode would mean 10^8 combinations, that would be impossible to guess in a minute.

4. Security Benefits

4.1 Effectiveness

Hard tokens have far more security than software tokens and its effectiveness will be discussed below, refer to Section 3.2 Soft vs Hard Tokens for details on soft token issues.

Hardware tokens are tamper resistant so it is almost impossible to duplicate a stolen token, and since they are physical items they cannot be 'internet hacked' or stolen without immediate detection. It also solves the problem of distributing a symmetric secret key over the internet, which would otherwise be done with asymmetric public and private keys which can easily be perfectly duplicated. Apart from a fatal weakness in the cryptographic implementation of the authentication code generation algorithm, which is very unlikely as it uses the extensively scrutinised AES-128 symmetric key encryption cipher, the only other way of compromise is if the token seed records are leaked from the RSA servers that store them.

RSA SecurID is also notably not security by obscurity, which by Kerckhoff's Principle that "If the cryptographic algorithm must remain secret in order for the system to be secure, then the system is less secure." Although the exact algorithm is propriety to the company and not publically release, it is based on the well-known and well-examined AES-128 encryption algorithm.

For corporations using the system, as the tokens are not internet enabled and seeds are stored on RSA servers, resulting in less internet connected endpoints that need to be protected. For corporations, they only need to protect the mapping of serial id numbers that identify each hard token to their respective user and for each user of the system they would only need to be responsible for their own unique hard token. RSA would be responsible for protecting the seed records on their own servers, which is much easier to achieve in their own controlled environment.

4.2 Attacks Protected Against

Although the following are attacks that SecurID offers protection against, it is important to note that these are not absolute and can still be hacked.

Attack	**Method of Protection**
Secret key stolen	Shared secret key is as safe as it can be in hard tokens which are factory embedded and set, this means the secret key is only known by the tamper resistant hard token and the RSA server that initialises it. The secret key is never transmitted over the internet unless the two factor authentication protocol is not properly adhered to.
Password spraying / brute force	One of the most common methods of attack where attackers use common passwords against a huge number of users, this makes company accounts, especially admin accounts, that are shipped with default passwords especially vulnerable. Since SecurID generates a unique random 6 - 8 digit authentication code every 60 seconds, this would become a pointless attack. It would be difficult for the attacker to compute 10^6 or 10^8 different combinations of digits every 60 seconds to brute force the authentication code.
Password replay	Attacker intercepts network communication and fraudulently delays or resends it to misdirect the receiver. This is protected against by the algorithm relying on a new authentication code every 60 seconds.
Password interception / man in the middle / perfect duplication	Attackers intercept the authentication credentials without detection and use it to log onto the system themselves. This would be extremely difficult as they would need to complete this in a 60 second time window and even then, the system is able to log an authentication attempt and restrict the authentication code to a one time use. SecurID authentication server tries to prevent password sniffing and simultaneous login by declining both authentication requests, if two valid credentials are presented within a given time frame.

5. Security Flaws

5.1 Ineffectiveness

Despite its security strengths, RSA SecurID still suffers from the same problem as traditional authentication which is that it relies on a shared secret that once stolen, can never be secret again. Although RSA SecurID has gone to lengths to protect the seed record which is only stored inside the token and stored inside their internal server, they suffered a brief in March 2011 where seeds were stolen from the company database. Once the seeds are stolen, if the attacker is able to map the seed to the corresponding user, then that user's identity can be faked.

It is important to note that hard tokens are factory embedded with the seed record and are designed to be tamper resistant, whereas soft tokens are initialised by entering a seed record into the application on the device. The initialisation of the soft token will inevitably introduce more vulnerabilities to the system since most seed records are sent over email by company admins with little to no encryption along with their corresponding serial id that is explicitly mapped to a user. If the seed record is intercepted, which often happens without detection, then that user's identity is easily faked. Refer to Section 3.2 Soft vs Hard Tokens for more details on soft token issues and how hard tokens mitigate the issues of soft tokens.

5.2 Attacks Vulnerable To

It is interesting to note that the attacks which are prevented are also often the attacks that the system is vulnerable to.

Attack	**Vulnerability**
Secret key stolen	Secret key is stored in the RSA servers which have been attacked successfully before, soft tokens are also much more likely hacked (internet connected) and does not offer tamper resistance of hard tokens.
Social engineering / phishing	Humans are the single point of failure in any system of security, and are prone to being tricked into sharing private information.
Password replay	Attacker intercepts network communication and fraudulently delays or resends it to misdirect the receiver. If the attacker is fast and manages to block the authorized user from authenticating to the server until the next token code will be valid, the attacker will be able to log in to the server instead of the intended user.
Physically losing the token	Physical items can always be lost as humans are the most error prone and vulnerable part of a system.

6. My Simplified Implementation

6.1 Outline of System

An insecure and simplified RSA SecurID implementation for learning purposes was created that implements the overall algorithm structure and time synchronisation functionality with dummy algorithms instead of full encryption implementation. System currently only supports one default user only.

Components

A two-factor authentication system for a user to a network resource (supports ONE user only). Uses an interval of 60 seconds to generate authentication codes, with leeway of +- one interval to account for drift to mimic hard tokens.

User: token
Mimic hard token generator that outputs an authentication code every 60 second interval. Can specify start_time.

Server: authenticate
Server that authenticates a user who is attempting to authenticate. Can specify current_time as the program doesn't use system time for ease of demonstration and learning purposes.

Usage

Note: user_id = serial_id = user_pin = seed = 1 for simplicity. Server assumes token to be initialised at start_time = 0.

To initialise and start token generator (user), run:

```
./token <serial_id> <seed> <start_time>
```

serial_id = id of token attached to user within rsa system
seed = secret symmetric shared key used for encryption
start_time = time when token initialised, default set to 0

To initialise and start authenticator (server), run:

```
./authenticate <user_id> <user_pin> <authentication_code> <current_time>
```

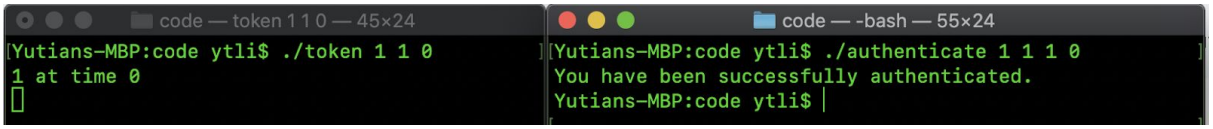
user_id = id of user within company system
user_pin = pin of user (personally set by user)
authentication_code = latest authentication code generated by token
current_time = time when authentication code is generated (explicitly set so unsynchronised tokens can be mimicked).

Sample Inputs and Corresponding Outputs

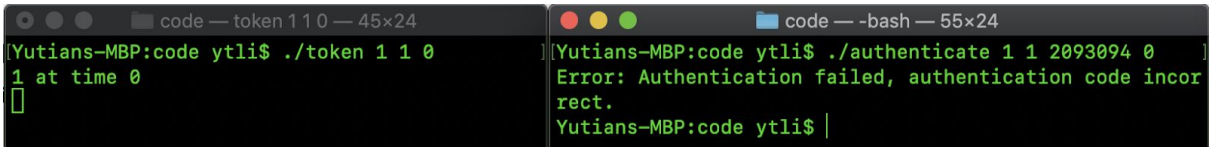
Note: user_id = serial_id = user_pin = seed = 1 for simplicity. Server assumes token to be initialised at start_time = 0.

Token Synchronised

Authentication code is entered correctly, authentication success.

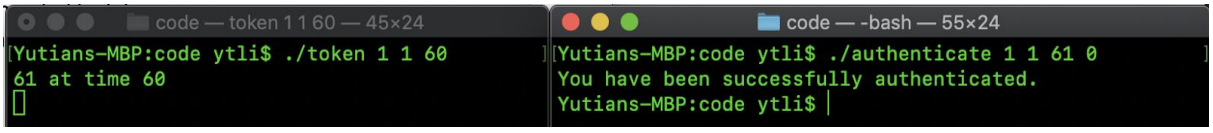


Random incorrect authentication code entered, authentication failed.



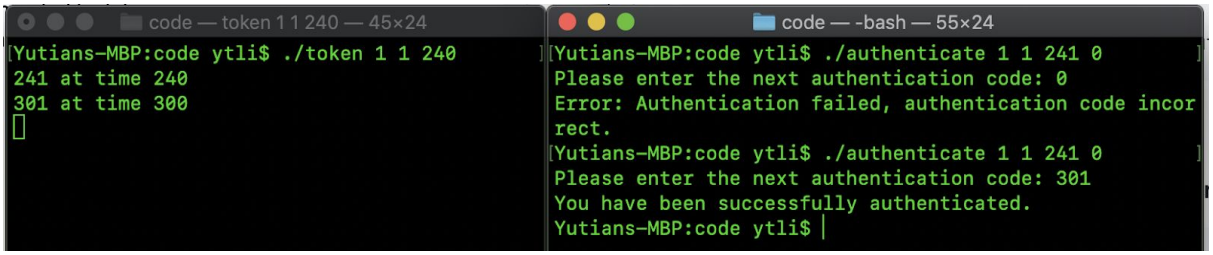
Token Not Synchronised (one interval time shift)

Mimic token drifting + one interval by setting start time of token to 60 seconds, token has drifted to 60 seconds faster than authentication server. Authentication code is counted as correct assuming time to have drifted as the algorithm allows for a ± one interval leeway. Algorithm first compares to authentication code generated at time 0, then checks for ± one 60 second interval.



Token Not Synchronised (multi-interval time shift)

Mimic token drifting + four intervals by setting start time of token to 240 seconds, token has drifted to 240 seconds faster than authentication server. Algorithm checks through ± all ten consecutive intervals to find a match, match found in the fourth interval, algorithm sends a challenge to the user for the next consecutive authentication code. First example shows a random code being entered and access being denied, this is to prevent an attacker from being authenticated after guessing one correct code. Second example shows the correct consecutive code being entered and authentication success.



6.2 Link to Github

https://github.com/ttian-yt/RSASecurID-Implementation