University of Stavanger

Faculty of Science and Technology

BACHELOR'S THESIS

Study program/ Specialization: Computer Science	Spring semester, 2023 Open / ~~Restricted~~ access
Writer: Oda Marie Johannessen	x [illegible] Oda Marie Johannessen
Faculty supervisor: Martin Gilje Jaatun External supervisor: Jacob Vetle Kvinnsland Ihle	
Thesis title: Bastion: An Application for Sharing Secrets	
Credits (ECTS): 20	
Key words: Cryptography, security, threat modeling, Azure, C#, Blazor, web application	Pages: 87 + enclosure: 13 Stavanger, 8 May 2023

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Bastion: An Application for Sharing Secrets

Bachelor's Thesis in Computer Science

by

Oda Marie Johannessen

Supervisors

Martin Gilje Jaatun
Jacob Vetle Kvinnsland Ihle

May 8, 2023

Abstract

The Bastion secret sharing solution is a web application that makes sharing secrets fast, easy and safe for employees and customers of Bouvet. It offers sharing encrypted secrets through one-click URLs with a limited lifetime. Once the secret lifetime is reached, or the secret URL is accessed, the secret is deleted from storage permanently.

When the user authenticates themselves in the web application, they can choose between one or more recipients of their choice to receive the secret, prior to it being encrypted and stored. A user accessing a secret URL where the sender has set predefined receivers will be prompted to log in to authenticate themselves. If they are one of the intended receivers, they will be able to view the secret and it's sender, and if not, they are denied access. If the URL somehow falls into the wrong hands, the secret is still protected by its predefined receivers.

If preferable, the secret can also be shared anonymously, and the user can choose not to authenticate themselves prior to creating and sharing a secret. The sender is not recorded along with the stored secret and no receivers are defined. The secret will be accessible for persons who are in possession of the secret URL and visit it before its lifetime is reached, or before anyone else has accessed the secret and it is deleted from storage.

Threat modeling has been carried out to uncover potential threats prior to the development phase of the web application, and the mitigation strategies found were utilized throughout the development process.

Acknowledgement

This thesis is a result of the cooperation between Bouvet and the University of Stavanger. The thesis has been completed as the final step of a Bachelor's in Computer Science.

I would like to thank both of my supervisors, Martin Gilje Jaatun and Jacob Vetle Kvinnsland Ihle, for all the help and guidance they have given me during this period.

Writing this thesis together with Bouvet has given me an unique opportunity to work on a task related to a real life problem, while observing and learning so much outside of the scope of my thesis. I am very grateful of Niklas Gustafsson who gave me the opportunity to pursue this thesis together with Bouvet.

I would also like to thank Simon Bay Andersen, who together with Jacob, took his time helping me and answering all my countless question.

Oda Marie Johannessen
Stavanger, 2023-05-08

Table of Contents

Nomenclature

Azure: Azure is Microsoft's cloud service platform [38].

Azure Active Directory (AD): Azure AD is Microsoft's identity provider [26]. The main responsibilities of an identity provider is storing and maintaining digital identities and providing authentication and authorization services for applications [26].

Azure AD Tenant: A tenant is an instance of Azure AD for a particular organization, and is separated from all other tenants in Azure AD [30].

Blob: Binary Large Object, storage solution which supports large amounts of unstructured data [27].

Domain-Driven Design (DDD): A development methodology used to manage system or application development, which follows strategic and tactical design concepts.

Globally Unique Identifier (GUID): A unique identifier following a special convention, which supports uniqueness across organizations and within an organization where the identifiers are created [43].

Key vault: A cloud service for storing and accessing secrets. To be able to perform operations on a key vault, you need to authenticate towards it [23].

Multi tenant: An application which is available for users registered in more than one Azure AD tenant [33].

Object ID (OID): A globally unique identifier (GUID) for a single user. The value is immutable [34].

OWASP: The Open Worldwide Application Security Project is a nonprofit foundation working towards improving software security [51].

OWASP Top Ten: A document for web application security which represents the current security risks deemed to be most critical for web applications [50].

Single tenant: An application which can only be used by users in one Azure AD tenant, the application's home tenant [33].

Storage account: A storage account in Azure contains different types of storage, including, among others, blobs, queues and tables [32].

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Sharing secrets

The need for sharing secret messages between a sender and receiver has existed for a long time, and it is today solved using cryptographic techniques. The goal is to share a secret as securely as possible, without compromising on the solution's user-friendliness. This combination can be difficult to achieve, and the most secure service will most likely not be the easiest to use.

As of today, Bouvet does not have a service that allows for sharing secrets with predefined receivers between employees and customers. Using Yopass [11], you can share encrypted secrets anonymously through a URL, but it is limited to small text files and short messages. Keeper [15] can be used internally between employees of Bouvet to share secret text files between groups, but cannot be used between employees and customers. This introduces the need for a solution which bridges the gap for sharing secrets between employees and customers of Bouvet.

The goal is to reduce, or avoid all together, the sharing of sensitive information like passwords or keys in plaintext to increase security, by introducing a quick and safe way of sharing secrets.

1.2 Motivation for a heightened security focus

The focus on security in software development has never been higher. One of the most recent, big breaches was that of LastPass in 2022, where a threat actor stole personal information of customers, including encrypted usernames and passwords and unencrypted website URLs [5]. Such a serious incident involving a large actor in password management [5] shows that no one is safe, and it demonstrates the need for secure coding practices and a high focus on risk analysis and threat mitigation.

OWASP's ranking of the most prominent security risks of today, the OWASP Top Ten list, mirrors this focus. The two top categories are broken access control and cryptographic failures, while the fourth place is taken by the category insecure design [50]. The top of list is dominated by risks related to authentication, authorization and cryptography.

1.3 The Bastion Secret Sharing Solution

The Bastion secret sharing solution is a web application, with supporting infrastructure in Azure, for sharing secrets which can be used by employees and customers of Bouvet. The application is end-to-end encrypted and easy to use. Prior to implementation of the sharing solution, a threat modeling has been performed to outline prominent risks and their mitigation strategies. The mitigation strategies have been utilized during development.

The primary functionality of Bastion is sharing secrets through a one-click link, with solid encryption/decryption logic. The messages are self-destructing and disappear after their lifetime has ended. The lifetime is chosen by the user, and has an upper limit of 24 hours. The user can also authenticate themselves by logging in to the application, and share the a secret one-click link with one or more predefined receivers of their choice. This provides the option of combining link lifetime and user authentication when sharing a secret. The users can decide who has access to a secret prior to sharing it, utilizing Azure Active Directory (AD) as the identity provider.

The Bastion secret sharing solution's web application can be visited at:

https://bouvet-bastion.azurewebsites.net

The secret sharing solution's web and Azure function application code can be viewed in the following GitHub repository:

https://github.com/odajohannessen/bastion

Chapter 2

Background theory

2.1 Cryptography

2.1.1 What is cryptography?

Cryptography is the discipline of providing secure communication in the presence of unauthorized third parties [42]. The need for two-way cryptographic methods arises when there is a message that needs to be sent by a sender to a receiver over an insecure channel. The insecure channel could be accessed by an adversary which the secret message must be protected from [9]. The main goal of cryptography is to protect sensitive data from being accessed, and subsequently read and/or altered by anyone other than the intended recipient [42].

The data is kept safe through the use of mathematical algorithms which convert the original data into a resulting message that can only be read if you have the appropriate tools to decode it [9]. A cryptographic method can also be one-way, which is commonly known as hashing, in which the resulting output cannot be decoded [40].

A two-way encryption scheme has the following setup: The content of the secret message the sender wants to communicate is known as the plaintext, and the message that is sent over the insecure channel is known as the ciphertext [9]. What separates the intended receiver and the adversary when handling the ciphertext, is that the receiver has some knowledge, a key, to use to transform the ciphertext into plaintext which can be read and understood [9].

The process of transforming plaintext into ciphertext is known as encryption [9]. An encryption algorithm uses an encryption key to convert the plaintext into a ciphertext [9]. The reverse process is known as decryption, and it also uses a decryption key to revert the ciphertext back to plaintext [9]. A two-way encryption scheme has been chosen because the purpose is to send and receive secret messages which need to be encoded and subsequently decoded.

2.1.2 Symmetric encryption and decryption

Symmetric encryption, also known as secret-key encryption, uses the same key for encryption and decryption [9], as demonstrated in Figure 2.1. To be able to use symmetric encryption for secure communication, the sending and receiving parties must first agree on a key (black) prior to sending any ciphertext over the chosen communication channel [9].

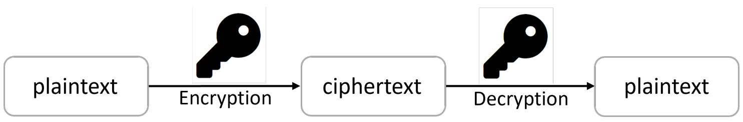


Figure 2.1: A simple sketch demonstrating symmetric encryption and decryption, redrawn from Kessler, G. C. [16]. The same key is used to encrypt the plaintext and decrypt the ciphertext.

The key can be shared between the parties in prior to communication using a key exchange [39]. The first key exchange protocol proposed is known as the Diffie-Hellman protocol [39]. Most encryption schemes use n-bit long keys, where n can be significantly smaller than the length of the plaintext in bits that is being encrypted [9].

A block cipher is a commonly used technique in symmetric cryptography [10], and can be used to encrypt data block by block [42]. A block cipher is a function which takes a key of length n bits and a plaintext message of length k bits as an input, and outputs a ciphertext of length k bits [10]. The two block ciphers currently approved for use by NIST are the Advanced Encryption Standard (AES) and the Triple DES [41]. AES is a 128-256 bit symmetric block cipher, with a key length set to either 128, 192 or 256 bits [6].

2.1.3 Asymmetric encryption and decryption

Asymmetric encryption, also popularly called public-key encryption, differs from symmetric encryption in that it uses two different keys for encryption (grey) and decryption (gold), shown in Figure 2.3. There is no need for a secure channel to first communicate the chosen key, and it is impossible to learn the decryption key from the encryption key [9].

The entities wishing to communicate with each other each use an algorithm to generate a key pair consisting of the encryption and decryption key, as demonstrated in Figure 2.2. The decryption key of each party is kept a secret, while the encryption key is published.

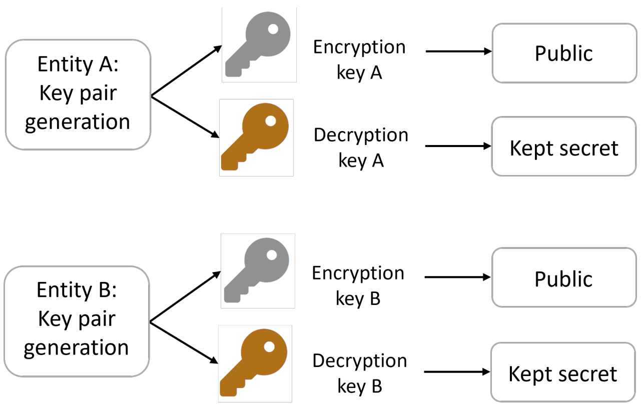


Figure 2.2: A simple sketch demonstrating key generation for asymmetric encryption and decryption, drawn from an example from Goldreich, O. [9].

Using the public encryption key of each entity, they can now communicate using each other's public encryption keys, and any ciphertext sent to either of the parties can only be decrypted by the party holding the correct secret decryption key [9], as demonstrated in Figure 2.3.

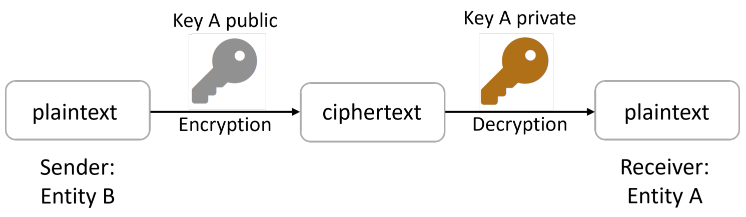


Figure 2.3: A simple sketch demonstrating communication between two entities A and B using asymmetric encryption, drawn from an example from Goldreich, O. [9].

2.1.4 One-way hash functions

A cryptographic hash function is a function which takes an input of arbitrary size and calculates a unique output of fixed size, called the digest [57]. Hash functions are called one-way functions because it is computationally infeasible to derive the input value from the digest itself. This feature is defined as being preimage resistant [57].

Another important property of hash functions are that they are second preimage resistant. This implies that it is difficult to find an input which can be hashed to become a specific output [57]. Hash functions are also collision resistant, meaning that two different inputs will not result in the same digest output [57].

If the input is changed slightly prior to hashing, the change in the output digest is significant. This is known as the avalanche property of the hash functions [57].

There are currently two hash function families approved for use by NIST, namely the SHA-2 and SHA-3 families [44]. One of the most common use cases for these one-way hash functions is password protection and storage [13].

2.2 OAuth 2.0, OpenId Connect and Azure AD

2.2.1 Authentication protocols

In OWASP's Top Ten web application security risks, we find identification and authentication failures as a prominent category [53]. The concept of authentication with regards to security means to verify the identity of a user, process or device [43]. The process of authentication is usually done prior to granting access to resources [43].

An authentication protocol can be defined as sequence of messages sent between a client and a service. This message sequence lets the service confirm the client's identity [43]. The authentication protocol transfers authentication data between the two entities [53].

A common authentication example is when a user authenticates itself towards a service by providing a username and password. These records can be checked against matching values in a database [53].

The following are the three factors that can be used to conduct authentication of a client [53]:

- Something you know
- Something you have
- Something you are

The first factor is most commonly a password, the second typically an SMS sent to your mobile phone containing a one-time password [1], and the third biometric information such as a fingerprint [53]. The third option, 'something you are', is the safest option of the three, and they are often used in combination to strengthen the authentication process [53].

2.2.2 Authorization protocols

Authorization is the concept of granting access or elevating privilege to an authenticated client [21]. In the state of being authorized, the client is given access to data, along with permissions of which operations can be performed on the data [21]. Authorization protocols handles this operation [21].

The permissions are often defined through role assignments, such as the common Administrator and User roles. Profiles can be assigned to these roles, and when they are, the profile obtains the authorization to complete actions and access resources as defined within these roles [37].

2.2.3 OAuth 2.0 Authorization Framework protocol

OAuth 2.0 is an industry standard protocol for authorization [52]. The main concept of OAuth 2.0 is to authorize and grant access without sharing any parts of the user's identity [12]. You can give a consumer access to private resources found at another service provider, without sharing your identity with said consumer [12].

An OAuth 2.0 flow defines the following roles [29]:

- Resource owner/user
- Resource server/API
- Client/application
- Authorization server

The client requests access to the resources owned by the user and hosted by a resource server. The server then issues an access token if approved by the user. The client can now use the access token to access the resources [28] [29].

The access tokens which are generated are primarely in JSON web token (JWT) format [29]. The permissions given by an access token are defined as scopes [29]. In OAuth 2.0 there are four primary grant types, or authorization flows, that provides access tokens [29]:

- Authorization code flow with/without PKCE
- Implicit flow with form post
- Resource owner password flow
- Client credentials flow (machine-to-machine)

Server-side applications can utilize the authorization code flow [47]. A prerequisite is that the source code of the application is not publicly exposed [47], since the application's client secret is a part of the flow [47]. The client secret is a secret used by the client/application to authenticate itself when sending requests to the authorization server [48].

The authorization code flow can be seen in Figure 2.4. The user chooses to log in when entering the web application (1). The process is redirected to the authorization server (2), and the server redirects the user to the login and authorization page or popup (3). The user logs in, and may or may not see a list of permissions they need to consent to when completing the login (4). After the successful login, the user is redirected back to the web application with an authorization code (5). The authorization code, client ID and client secret is sent to the authorization server (6), where these are verified (7). After a successful validation, the server sends back an access and refresh token (8), which the application can use to call the API to access information about the user (9). The API then responds with the user data (10).

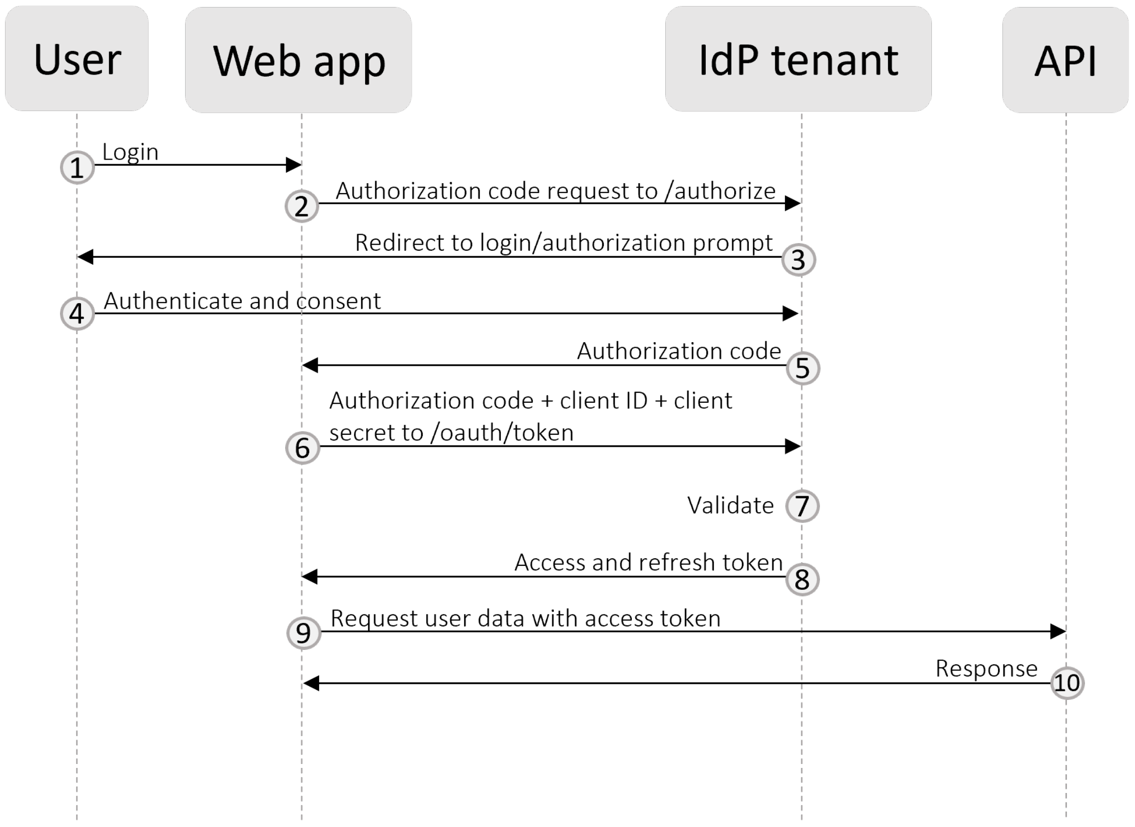

User
Web app
IdP tenant
API
1 Login
2 Authorization code request to /authorize
3 Redirect to login/authorization prompt
4 Authenticate and consent
5 Authorization code
6 Authorization code + client ID + client secret to /oauth/token
7 Validate
8 Access and refresh token
9 Request user data with access token
10 Response

Figure 2.4: A sketch demonstrating the OAuth 2.0 authorization code flow, redrawn from Auth0 [47]. The user is the resource owner, the web application the client, the IdP tenant the authorization server, and the API the resource server.

2.2.4 OpenId Connect (OIDC) protocol

The OpenId Connect protocol is an identity layer on top of the OAuth 2.0 framework [49]. In addition to the flows offered by OAuth 2.0, it provides third parties the ability to authenticate users, and to obtain basic information of said users [49]. The JSON web tokens (JWTs) are used by OIDC for issuing tokens when a user is authenticated [49], and the authentication code flow can be seen in Figure 2.5. This flow differs from the OAuth 2.0 in step (8) described above, where the authorization server returns an ID token in addition to the access token.

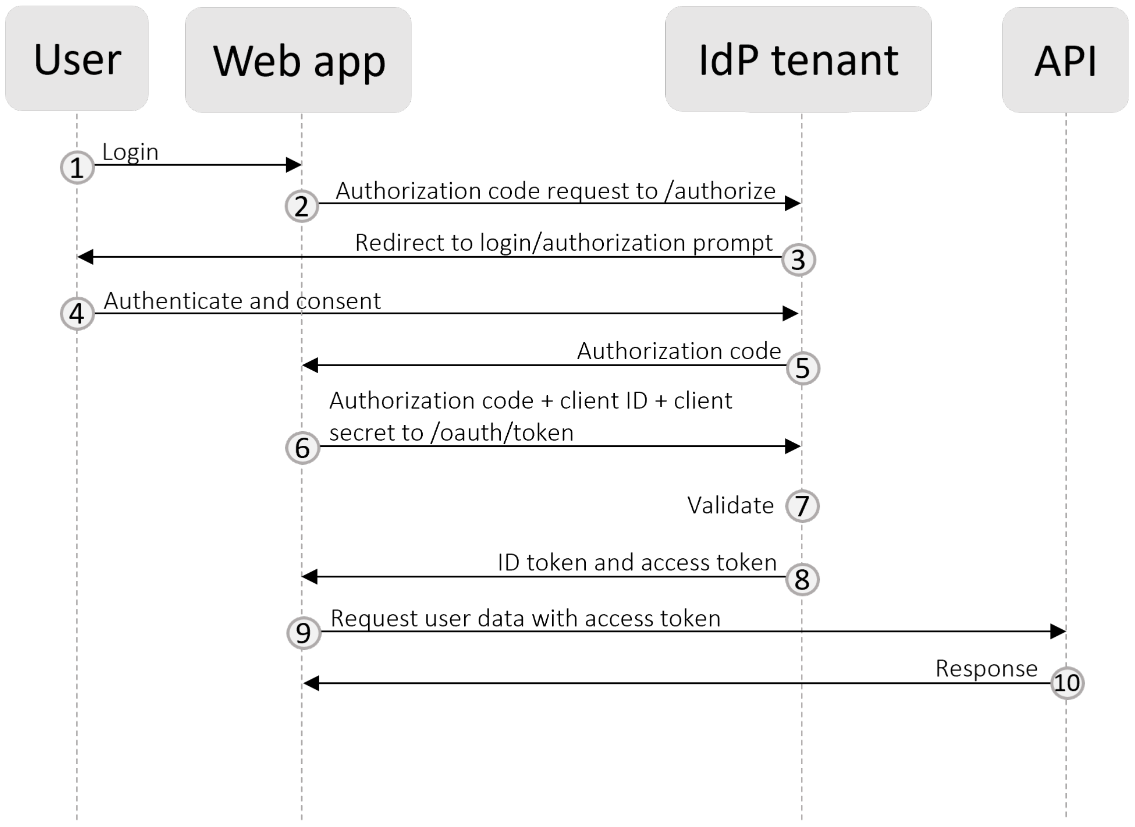

User
Web app
IdP tenant
API
1 Login
2 Authorization code request to /authorize
3 Redirect to login/authorization prompt
4 Authenticate and consent
5 Authorization code
6 Authorization code + client ID + client secret to /oauth/token
7 Validate
8 ID token and access token
9 Request user data with access token
10 Response

Figure 2.5: A sketch demonstrating the OAuth 2.0 authorization code flow with OIDC, redrawn from Auth0 [47]. The user is the resource owner, the web application the client, the IdP tenant the authorization server, and the API the resource server.

2.2.5 Identity Provider (IdP)

The main responsibilities of an identity provider is storing and maintaining digital identities and providing authentication and authorization services for applications [26]. An identity provider is a trusted entity that issues credentials to either its subscribing applications or itself following authentication [45].

An identity provider can either be self-hosted, such as Keycloak [17], or external like Azure Active Directory [26]. The motivation for using an externally managed identity provider, instead of managing user credentials and information on your own, is to lessen the administrative burden [21].

In the authentication flow in Figure 2.4 and 2.5, the identity provider is the tenant which issues access, refresh and ID tokens.

2.2.6 Azure Active Directory (AD)

Azure Active Directory, also called Microsoft identity platform, is Microsoft's identity provider [26]. Azure AD supports several protocols for authentication and authorization, with OAuth 2.0

and OpenId Connect being two of them [22]. Azure AD supports all OAuth 2.0 flows [28]. In Figure 2.4 demonstrating the OAuth 2.0 authorization code flow, Azure AD is the IdP tenant, or authorization server. The Azure AD account is the identity used in the authentication process [35].

In the case of web applications, which utilize the authorization code flow, the authentication and authorization is done using the OAuth 2.0 framework and the OpenId Connect protocol, with Azure AD as the authorization server which issues tokens [21].

A tenant is an instance of Azure AD for a particular organization, and is separated from all other tenants in Azure AD [30]. The object ID is a globally unique identifier (GUID) which exist for each user registered in an Azure AD tenant [34]. The value is read-only and immutable [34].

2.2.7 Application registration

An application registration is a registration of a particular application within an Azure AD tenant [19]. The application registration is necessary to be able to delegate access and roles [19]. The application registration can either be set as single or multi-tenant, and it represents a globally unique instance of the registered application [19].

When the application registration is created in a tenant, an application and service principal object are also constructed. The application object resides in the tenant in which the application registration was created [19]. The application object can be used to create service principal objects [19], and a service principal will exist in every tenant that the application is used [19].

The application object defines how tokens are issued to be able to access the application, any resources it can access and actions the application is allowed to perform [19]. The application object is global [19]. The service principal is necessary to enable the ability for users to sign in to the application, and to give access to Azure resources [19].

2.3 Threat modeling

2.3.1 What is threat modeling?

Threat modeling is the process of identifying and mitigating threats with regards to an application or a system [7]. Its purpose is to make potential threats obvious at an early stage, and to determine a mitigation strategy to following during the life cycle of an application [7].

A threat model can contain the following steps, as defined by OWASP [7]:

- **Security goals and requirements**: Assumptions regarding the security of the application or system
- **Decomposition of the application**: A description of the application

- **Determine and rank threats**: Potential threats that are identified
- **Determine countermeasures and mitigation**: Mitigation strategies for each of the identified threats

The threat model should be updated and refined as new features are added to the application during development, technical decisions are made, or in the case of relevant security incidents [7]. The benefits of threat modeling in decision making is that its threats and mitigation strategies can be used as specific arguments for security decisions being made. It justifies using time and resources to implement security measures [7].

2.3.2 Security goals and requirements for an application

Software security defines the three following traditional goals for any application or system [54]:

- **Confidentiality**: Protecting secret or private information from unauthorized information release
- **Integrity**: Protecting information from unauthorized modification
- **Availability**: Legitimate users should not be denied access to an application or a system

Some other important security goals which should also be met are [54]:

- **Authenticity**: The quality of being original, not tampered with
- **Non-repudiation**: The inability to deny having performed an action

The security goal itself describes a wanted property of the software [3], while a security requirement describes some functionality needed to reach a specific security goal [3].

2.3.3 Decomposition of the application

By creating a detailed description of the application, the persons participating in the threat modeling gain an understanding of how the application works, or will work, and how it communicates with external entities [4]. The description can be created by identifying the following [4]:

- Use and misuse cases
- External dependencies
- Entry and exit points
- Application assets
- Trust levels
- Data Flow Diagrams (DFD)

Generating use and misuse cases

Use cases are commonly used to demonstrate functional and security requirements expected of the application [18]. A simple sketch demonstrating a use and misuse case can be seen in Figure 2.6.

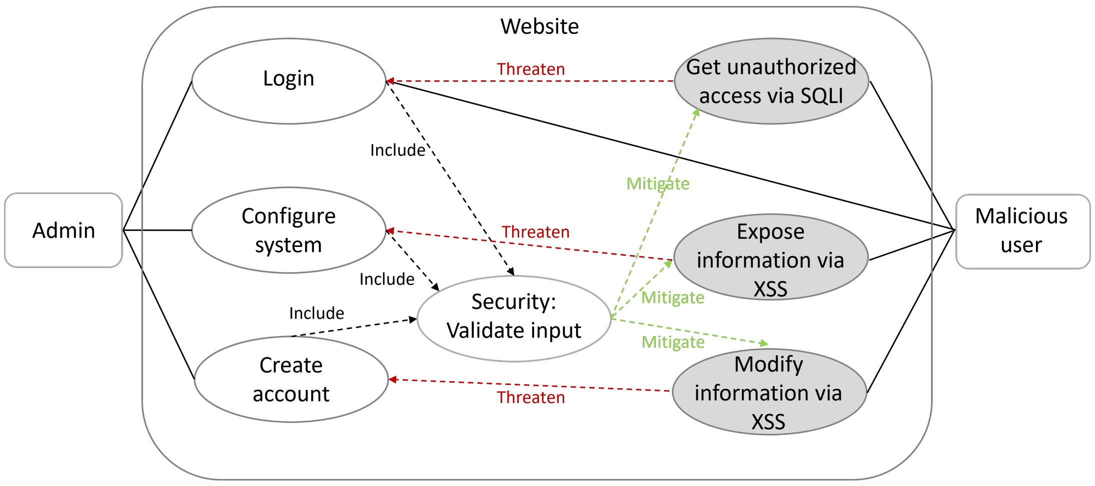


Figure 2.6: An example of a use and misuse case diagram, redrawn from Mai, P. X. et al. [18].

Identifying external dependencies

An external dependency can be defined as anything outside of the code itself which could be a potential threat to the application [4].

Identifying entry and exit points

An entry point, also known as an attack surface, is anywhere in the application a malicious user or an attacker can interact with the application by providing input data [4]. An exit point is anywhere in the application where data or responses are returned to the user [4]. An entry point combined with an exit point encloses a trust boundary [4].

Assets of the application

The assets of an application is simply anything of value an attacker might wish to gain access to and/or exploit [4].

Trust Levels

A trust level, made up of trust boundaries, defines which access rights are given to different entities outside of the application. A trust level is connected by an entry and an exit point [4].

Data Flow Diagram

The purpose of a data flow diagram is to display how data is processed, and where it is sent, for different processes or scenarios [4]. An example of a data flow diagram, along with an explanation of the symbols used, can be seen in Figures 2.7 and 2.8, respectively.

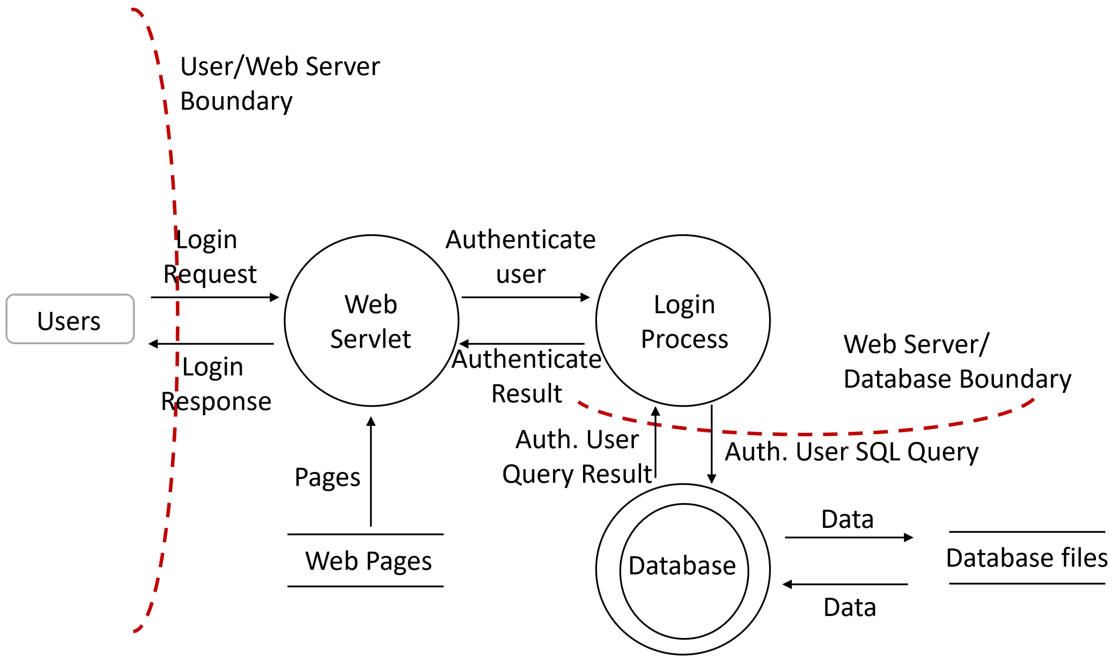


Figure 2.7: An example of a data flow diagram (DFD), redrawn from the OWASP Threat Modeling Process [4].

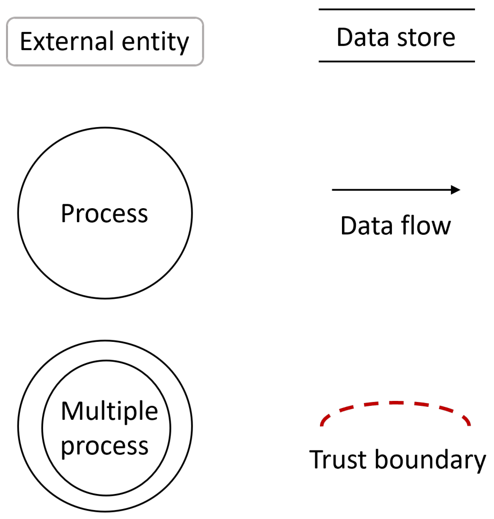


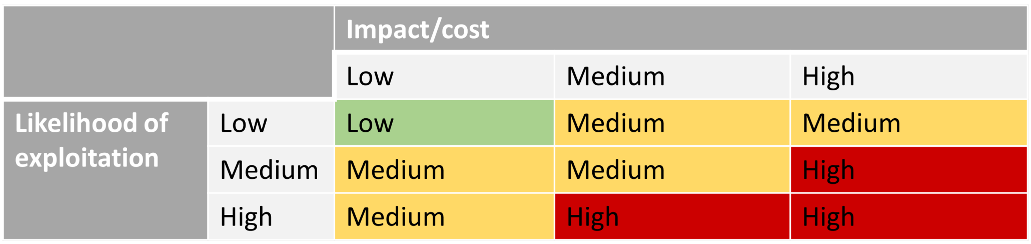
Figure 2.8: Data flow diagram symbol description, redrawn from the OWASP Threat Modeling Process [4].

2.3.4 Determine, categorize and rank threats

STRIDE is one of several threat categorization tools, and it utilizes a list of generic threat categories that can be used to identify potential threats. The categories are as following [4]:

- **Spoofing**: A malicious actor pretending to be someone they are not
- **Tampering**: A malicious actor modifying data
- **Repudiation**: A malicious actor can deny having performed (illegal) actions because of lack of traceability
- **Information Disclosure**: A malicious actor reading files they should not have access to
- **Denial of Service**: A malicious actor denying access to a service for legitimate users
- **Elevation of Privilege**: A malicious actor gaining a privilege level they should not have authorization to achieve

After using the STRIDE mnemonic to determine the potential threats that are present, the next step is to rank them based on risk. For simplicity, a simple matrix will be used to do the risk calculation [8]:



		Impact/cost		
		Low	Medium	High
Likelihood of exploitation	Low	Low	Medium	Medium
	Medium	Medium	Medium	High
	High	Medium	High	High

Figure 2.9: A simple risk matrix used to rank threats based on likelihood of exploitation and potential impact. Redrawn from Duijm, J. D. [8].

The potential threats can be ranked by low, medium or high risk depending on the likelihood of occurrence and the potential impact or cost [4].

Determine countermeasures and mitigation strategies

Countermeasures and mitigation strategies can be set up for each potential threat in the different STRIDE categories [4]. Sorting the threats and mitigation strategies by risk, this overview can be used as a decision making tool when choosing which mitigation strategies to prioritize [4].

2.4 Domain-Driven Design

2.4.1 What is Domain-Driven Design (DDD)?

Domain-driven design is one of many development methodologies that can be used to manage the development of a system or an application [55]. Domain-driven design follows the concepts of strategic and tactical design [56].

Strategic design, or modeling, defines the domain model and its content [56]. Through strategic design, subdomains and bounded contexts are created, an ubiquitous (familiar) language is decided for each bounded context, and subdomains are assigned to them [56]. An example of a domain model as a result of strategic design can be seen in Figure 2.10. An ubiquitous language is a language everyone involved in the development and decision making of a software understands [56].

The next step after strategic design is the tactical design [56]. Through tactical design it is decided how the domain model should be implemented, by defining entities and value objects, and building aggregate roots [56], as seen in Figure 2.11. Following the tactical design, the software is implemented to reflect the domain model itself [56].

Some of the main advantages to DDD are that the resulting domain model and implementation of it demonstrates how software will work [56], and that domain experts, decision takers and developers understand each other using the same familiar language [56].

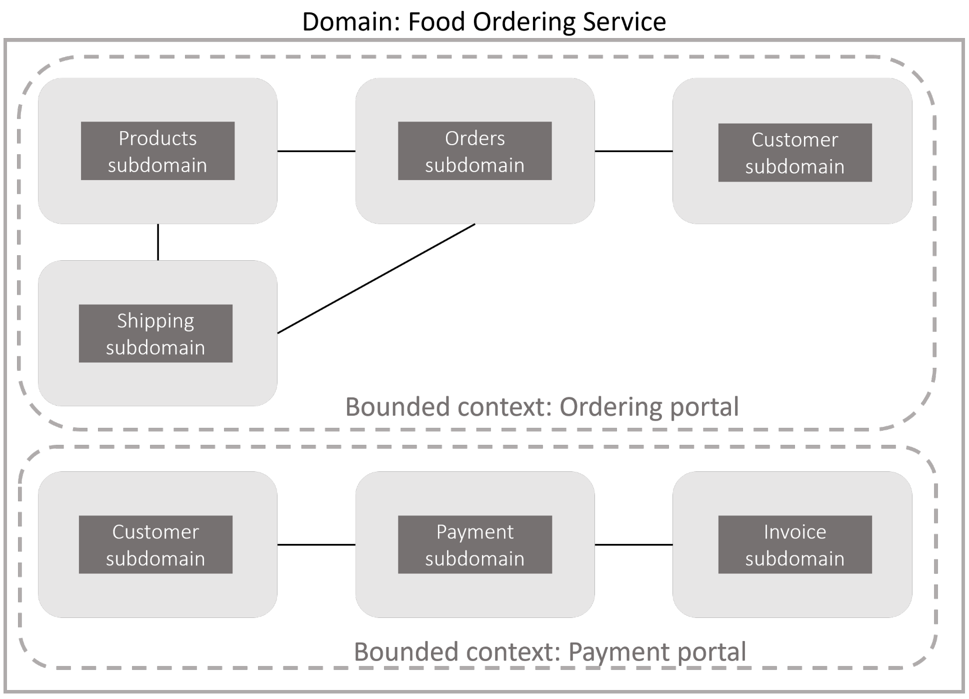


Figure 2.10: A simple sketch of a domain model, redrawn from Vaughn, V. [56]. Each bounded context uses an ubiquitous language, and the connections between subdomains demonstrate the relationships between them.

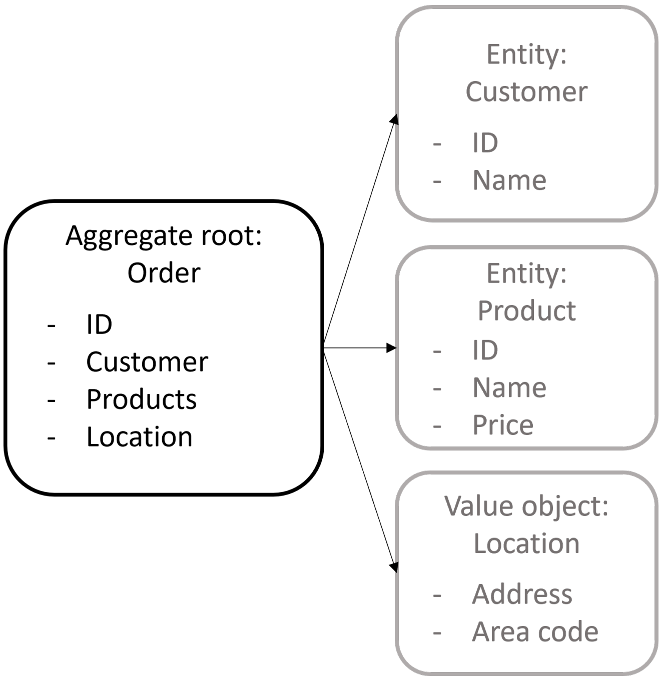


Figure 2.11: A simple sketch of an aggregate root, redrawn from Vaughn, V. [56]. The aggregate root Order has two entities and one value object associated with itself.

Requirements for a domain model

A domain model is a simplified version of reality. The code itself should be a representation of the domain model in practice [14]. A good domain model needs to be simple, capture the essence of the functionality of the system, and use a familiar language that everyone involved understands [14] [56]. A domain model can represent parts of a business logic, or a whole system [56].

2.4.2 How can DDD be combined with software security?

If one knows exactly what a system should be doing, it is implicitly known what it should *not* be able to do [14]. The expected behaviour of a system is well known after performing strategic and tactical modeling [56]. The domain model can be used as an asset when decomposing the application or system during threat modeling. It can also be used to define the trust boundaries and entry and exit points, because the DDD domain model provides us with a separation of concerns [55].

Chapter 3

Threat Modeling of the Secret Sharing Solution

3.1 Security goals and requirements

- **Confidentiality**: Keeping the user's secret private throughout the lifetime of the secret.
- **Integrity**: The secret that is stored should not be modified or deleted prior to end of lifetime by a party not authorized to perform the action.
- **Lifetime of a secret chosen by user should be upheld and respected**: The secret should be deleted completely and not be available to anyone once its lifetime is reached.
- **A secret should be deleted once its URL is accessed**: Once the URL has been utilized by a recipient, the secret should be deleted completely and not be available to anyone, not even the sender.
- **Availability**: The application should be available anytime a user wishes to create a secret, or view a secret by accessing a URL, with as little downtime as possible.
- **Non-repudiation for anonymous users**: Logging of activity by documenting creation of secrets, accessing secrets and deleting secrets.
- **Non-repudiation for authenticated users**: Logging will document the individuals who create and access secrets.
- **Authenticity for anonymous users**: The identity of the individuals who create or access secrets will not be known.
- **Authenticity for authenticated users**: The identity of the individual who create or access secrets will be known if the user chooses to authenticate themselves prior to creating a secret and selects receiver(s) for the secret.

3.2 Decomposition of the secret sharing solution

3.2.1 Use cases for anonymous users

Three use cases have been created for anonymous users utilizing the secret sharing solution to define its desired basic functionality, as seen in Figures 3.1-3.3 below. The use cases have been created using the domain model in Figure 4.1.

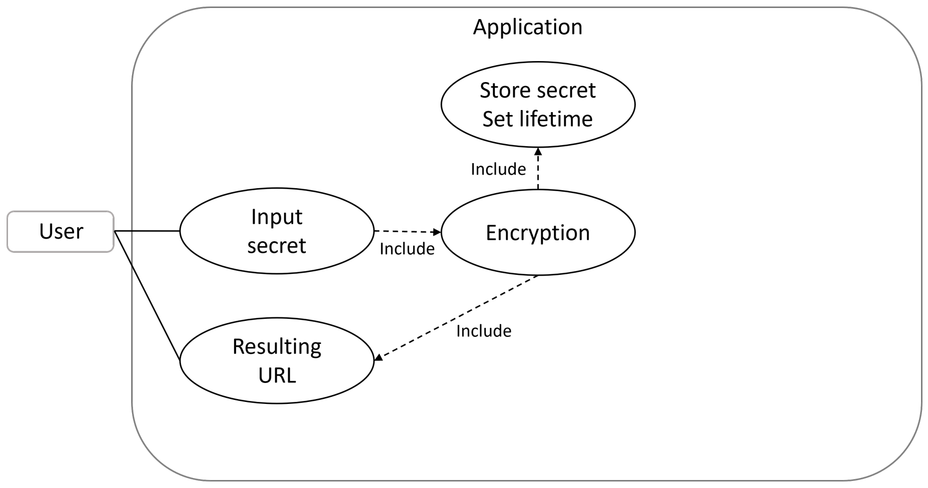


Figure 3.1: The use case of an anonymous user creating a secret, setting its lifetime and receiving the secret URL.

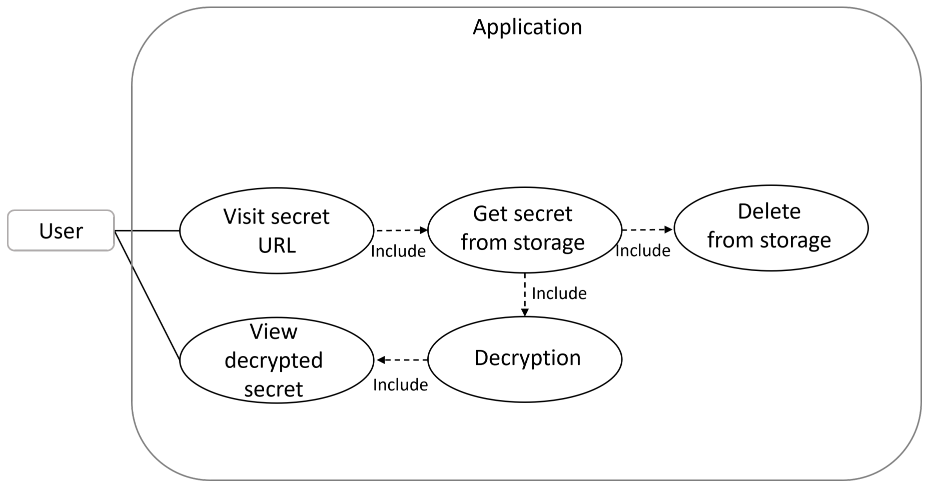


Figure 3.2: The use case of an anonymous user accessing a secret through a secret URL. The secret is deleted from storage once the link is accessed.

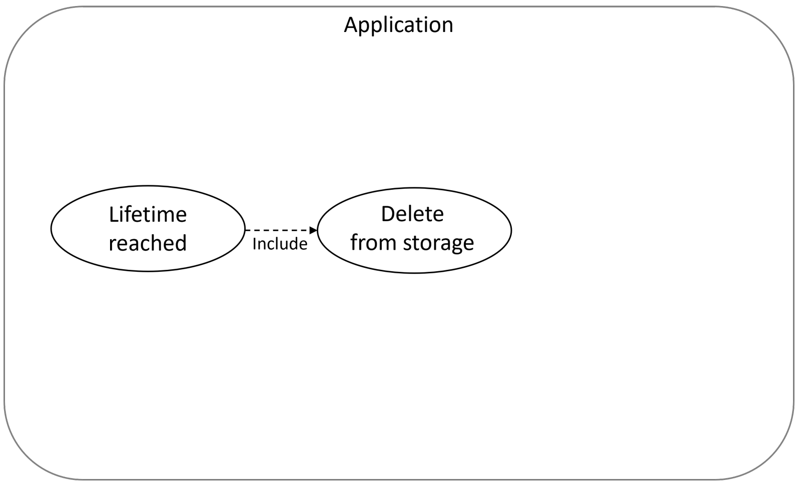


Figure 3.3: The use case of a secret exhausting its lifetime before being viewed, when the secret is deleted from storage.

3.2.2 Use cases for authenticated users

Three use cases have been created for authenticated users utilizing the secret sharing solution, as seen in Figures 3.4-3.6 below. The use cases have been created using the domain model in Figure 4.2.

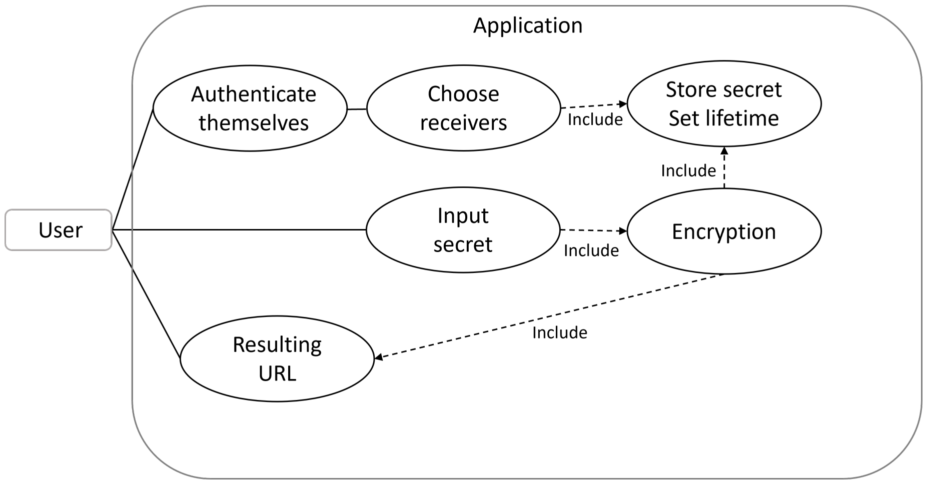


Figure 3.4: The use case of an authenticated user creating a secret, setting its lifetime and receiving the secret URL.

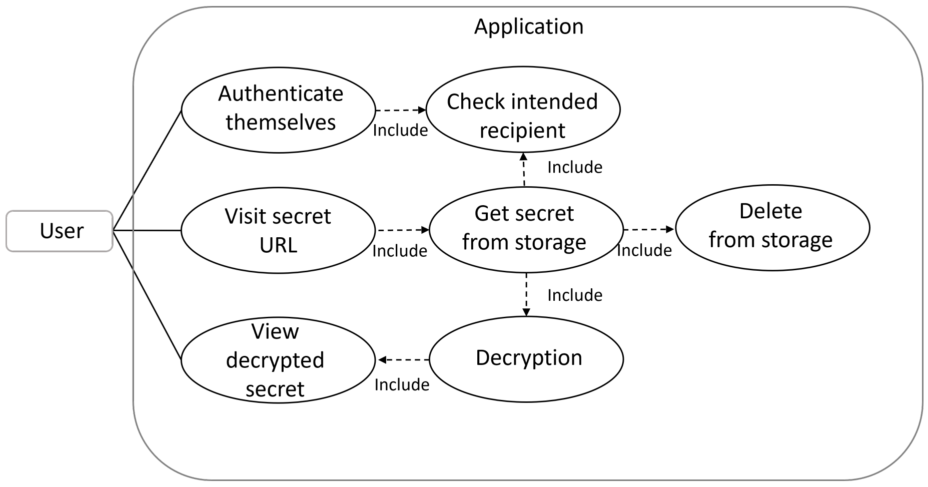


Figure 3.5: The use case of an authenticated user accessing a secret through a secret URL. The user is granted access to the secret if they are one of the intended receivers, and the secret is deleted from storage when all the receivers have viewed the secret.

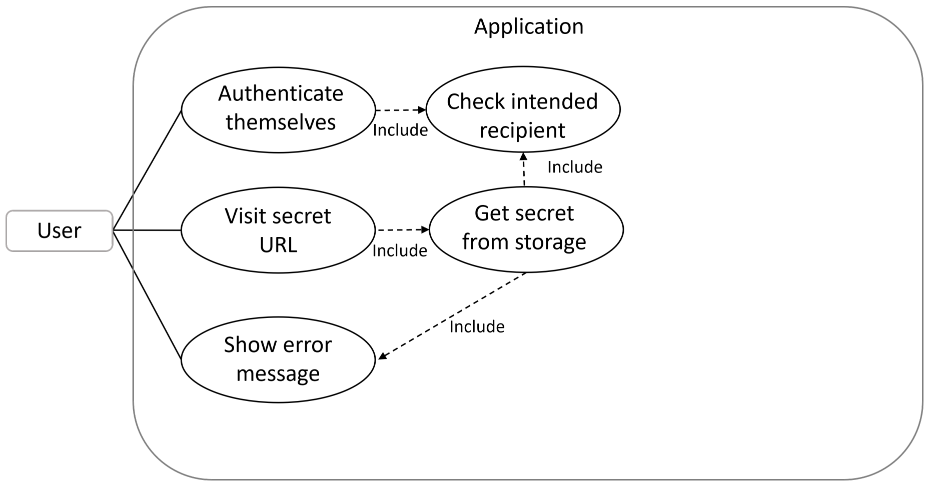


Figure 3.6: The use, or abuse, case of a user trying to access a secret where they are not the intended receiver.

3.2.3 External dependencies

The application's external dependencies can be seen inn Table 3.1. A visualisation of the Azure resources can be seen in Figure 4.7.

External dependency	**Description**
Azure resources	The solution will be running as a set of Azure resources
Secret storage	Azure blob storage and key vault
Connection between application and data store	Internal connection between blob storage, key vault and web application
Blazor Server-side Framework	External framework and libraries of code utilized
External libraries	External libraries of code utilized
Azure AD tenant	The tenant where the application is registered and granted permissions
Application registration	The registration of the application in the Azure AD tenant
Microsoft OAuth 2.0 authorization and OpenId Connect authentication flow	Microsoft.Identity.Web external library and redirection to Microsoft authentication portal

Table 3.1: An overview of the external dependencies for the secret sharing solution.

3.2.4 Entry and exit points

The secret sharing solution will have the following entry and exit points, as defined in Table 3.2.

Type	Name	Description	Trust Level
Entry	HTTPS port	Entry point for connecting to the application through a network port	Anonymous or authenticated user
Entry	Application index page	Entry point for secret input	Anonymous or authenticated user
Entry	Application index page	Entry point for defining receivers	Authenticated user
Exit	Application index page	Exit point for viewing secret URL	Anonymous or authenticated user
Exit	Application show secret page	Exit point for viewing secret	Anonymous or authenticated user
Exit	Application show secret page	Exit point for viewing sender of secret	Authenticated user
Entry	Application login redirect	Entry point for credentials	Authenticated user
Exit	Application logout redirect	Exit point redirecting to index page	Authenticated user

Table 3.2: An overview of the entry and exit points in the secret sharing solution.

3.2.5 Assets of the application

The assets of the application are the encrypted secrets and the Azure object ID of the sender and receiver(s) if the user has authenticated themselves. This ID is unique for all Azure AD accounts [34].

Name	Description	Trust level
Encrypted secrets	Secrets stored in Azure blob storage	Anonymous or authenticated user with secret URL
Object ID (OID)	OID of sender and receiver of secret stored along encrypted secret in Azure blob storage	Application

Table 3.3: The assets found in the secret sharing solution.

3.2.6 Trust levels

The secret sharing solution has two trust levels, the level of the user, and the application level.

- Create secret: Anonymous or authenticated user - user/web server boundary
- View secret: Anonymous or authenticated user with URL - user/web server boundary
- Store, retrieve or delete secret: Application - Web application/Azure storage or key vault boundary

3.2.7 Data Flow Diagrams for anonymous users

The data flow diagrams are shown in Figure 3.7 and Figure 3.8. The trust levels defined above can be seen as boundaries in each figure. The data flow diagrams have been created using the domain model in Figure 4.1.

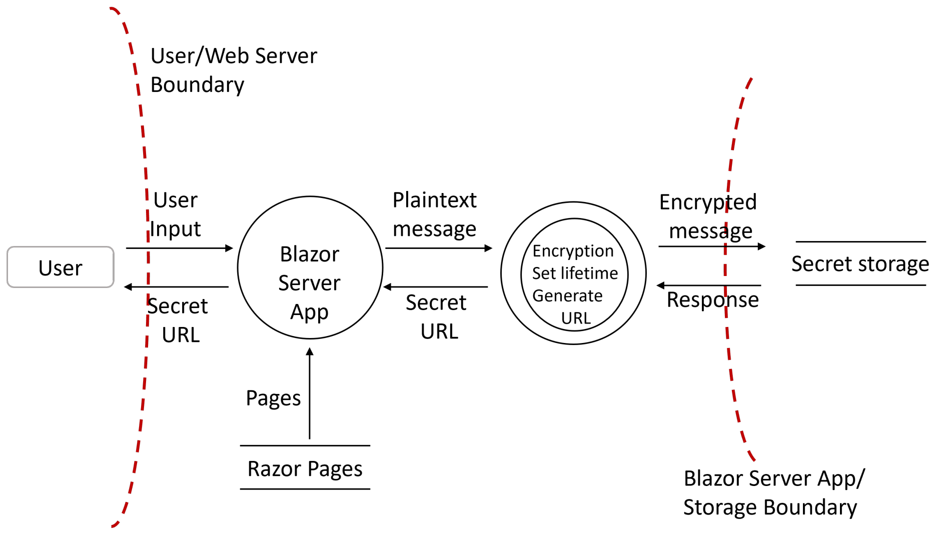


Figure 3.7: The data flow diagram (DFD) of the process of an anonymous user creating a secret.

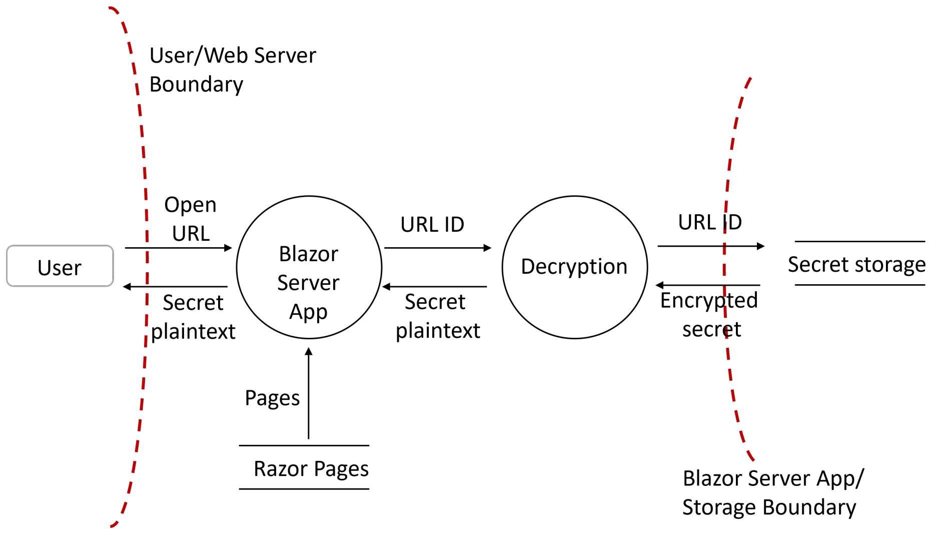


Figure 3.8: The data flow diagram (DFD) of the process of an anonymous user accessing a secret.

3.2.8 Data Flow Diagrams for authenticated users

The data flow diagrams for authenticated users are shown in Figure 3.9 and Figure 3.10.

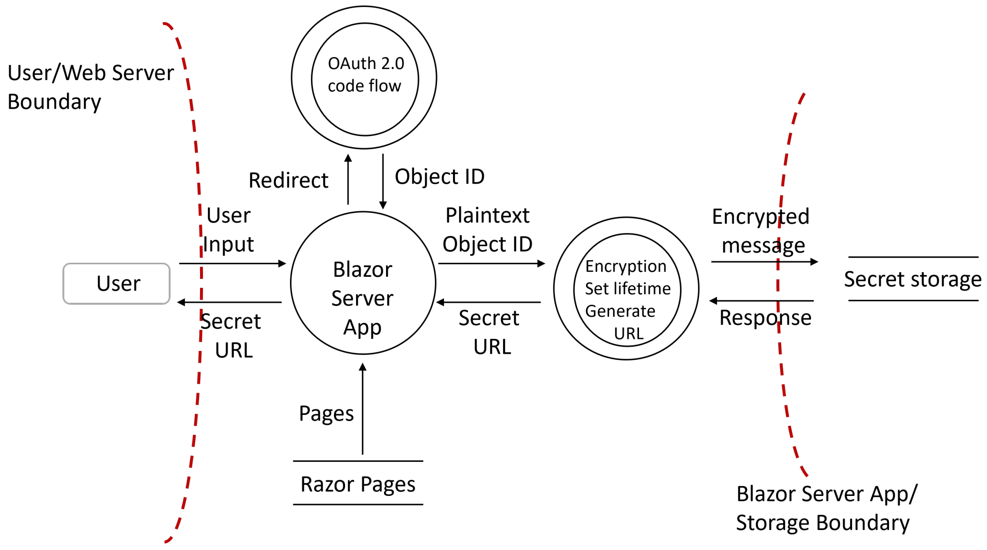


Figure 3.9: The data flow diagram (DFD) of the process of an authenticated user creating a secret.

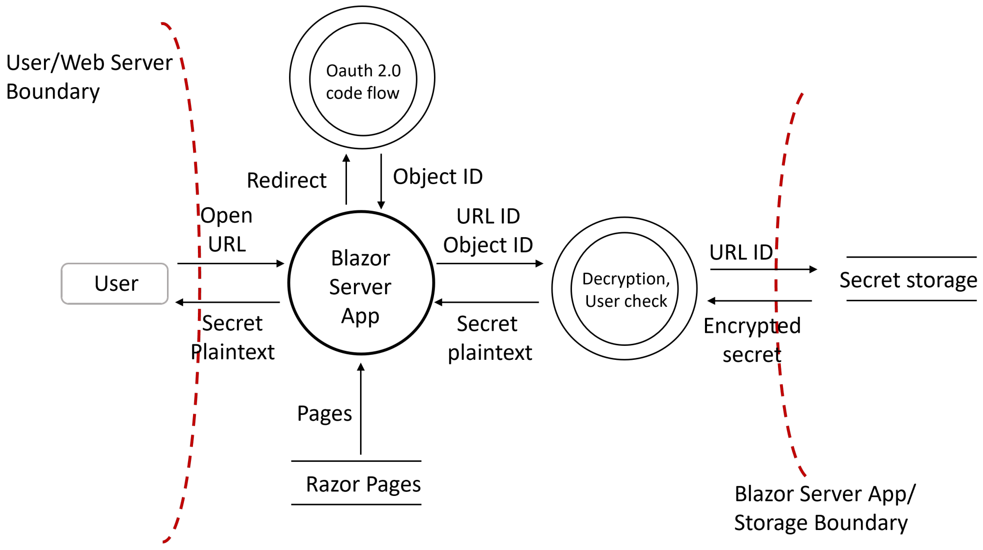


Figure 3.10: The data flow diagram (DFD) of the process of an authenticated user accessing and reading a secret.

3.3 Identified threats and mitigation strategies

3.3.1 STRIDE analysis and ranking of threats

The potential threats defined using STRIDE and their ranking based on low to high risk can be seen in Table 3.4. The risks have been calculated using the table in Figure 2.9.

STRIDE Category	**ID**	**Potential threats**	**Risk**
Spoofing	1	Identifying as a different user and creating a secret using their credentials	Medium
	2	Accessing a secret without being the intended recipient	High
Tampering	3	Altering stored secrets without permission	High
	4	Altering stored secrets by injection	High
	5	Altering lifetime of stored secrets without permission	Low
	6	Altering lifetime of stored secrets by injection	Low
	7	Altering receiver(s) of secret	Medium
Repudiation	8	Altering stored secrets without it being traced	High
	9	Altering lifetime of stored secrets without it being traced	Low
	10	Altering receiver(s) of secret without it being being traced	Medium
Information Disclosure	11	A threat actor accessing and reading stored secrets without a valid URL	High
Denial of Service	12	The application becoming unavailable due to a DoS attack	High
Elevation of Privilege	13	Gain privileged access to secret storage or a secret where the user is not the intended recipient	High

Table 3.4: STRIDE threat identification and ranking of the possible risks found for the secret sharing solution.

The risk calculation for each threat can be seen in Table 3.5.

ID	**Probability of occurrence**	**Impact**	**Risk**
1	Medium	Medium	Medium
2	Medium	High	High
3	Medium	High	High
4	Medium	High	High
5	Low	Low	Low
6	Low	Low	Low
7	Medium	Medium	Medium
8	High	Medium	High
9	Low	Low	Low
10	Medium	Medium	Medium
11	Medium	High	High
12	Medium	High	High
13	Medium	High	High

Table 3.5: Calculation of risk of each individual threat. The risk is defined as the likelihood of occurrence multiplied with impact.

3.3.2 Mitigation strategies

The mitigation strategies for each initial risk, identified by ID, can be seen in Table 3.6. The residual risk is the estimated remaining risk of a threat occurring after the mitigation strategy has been applied.

ID	Initial risk	Mitigation strategy	Residual risk
1	Medium	Utilize OAuth 2.0 and OIDC authentication flow for user authentication	Low
2	High	Utilize Azure AD as the IdP and outsource authentication, token validation	Low
3	Medium	Using a separate key vault for storage of symmetric key, apart from the blob storage for secrets	Medium
5	Low	Using a separate key vault for storage of symmetric key, apart from the blob storage for secrets	Low
7	Medium	Token validation, only administrators can access the Azure AD tenant (to be able to access user OID)	Low
8	Medium	Logging	Low
9	Low	Logging	Low
10	Medium	Token validation, only administrators can access the Azure AD tenant (to be able to access user OID)	Low
11	High	Access to blob storage managed through managed identity, store key in key vault	Medium
12	Medium	Azure offers DDoS protection for deployed resources [36]	Low
13	High	Using a separate key vault for storage of symmetric key, apart from the blob storage for secrets	Medium
4	Medium	Using blob storage and key vault instead of a SQL database, code won't be executed	None
6	Low	Using blob storage and key vault instead of a SQL database, code won't be executed	None

Table 3.6: Identified mitigation strategies for each threat.

Other general mitigation strategies:

- Use the latest available update of external libraries
- Stay updated on Blazor framework and external library vulnerabilities
- Utilize Microsoft managed identity to avoid having to use secrets to connect to the storage account, the key vault or to perform logging
- Follow advice from Microsoft in case of incidents related to Azure

Chapter 4

Implementing the Secret Sharing Solution

4.1 Technical decisions

Type	**Decision**	**ADR**
Threat modeling process	OWASP and STRIDE mnemonic	A.1
Security requirements	As defined in Section 3.1	A.2
Cloud service provider	Microsoft Azure Cloud Services	A.3
Identity provider	Azure Active Directory (AD)	A.4
Location of hosting of resources	Norway	A.5
Programming language	C#	A.6
Software framework	.NET	A.7
.NET version	6	A.8
Web application framework	Blazor Server-Side	A.9
Request/Response messages	MediatR library	A.10
Unit testing framework	XUnit	A.11
Cryptographic algorithm	AES	A.12
Storage solution	Storage account and key vault	A.13
Azure role assignment for storage	User-assigned Managed Identity	A.14
Basic functionality	One-click link with chosen lifetime	A.15
Domain model	The domain model in Figure 4.1	A.16
Limit of lifetime of secrets	24 hours	A.17
Deletion of expired secrets	Azure function application	A.18
Random and unique identifier	ID and URL generation using C# GUID class	A.19
Host Blazor web application	Host as an Azure web Application	A.20
Deployment type	Deploy web application from Visual Studio	A.21

Table 4.1: Part 1: Technical decisions made during the thesis work.

Type	**Decision**	**ADR**
Authorization protocol	OAuth 2.0 framework	A.22
Authentication protocol	OpenId Connect (OIDC)	A.23
Expanded functionality	One-click link with predefined receiver(s)	A.24
Logging	As defined in Section 4.9.6	A.25
AD tenant	An AD tenant has been created for testing purposes	A.26
Application registration roles	As defined in Section 4.8.2	A.27
ID for senders/receivers	Immutable and unique OID	A.28
Token validation	Validate audience and issuer	A.29
Storage of receivers	Key-value pair of OID and viewer status in blob metadata	A.30

Table 4.2: Part 2: Technical decisions made during the thesis work.

For more information, see the architectural design record (ADR) in Appendix A for details surrounding each decision.

4.2 Domain model and aggregate roots

The domain model of the secret sharing solution for anonymous users can be seen in Figure 4.1.

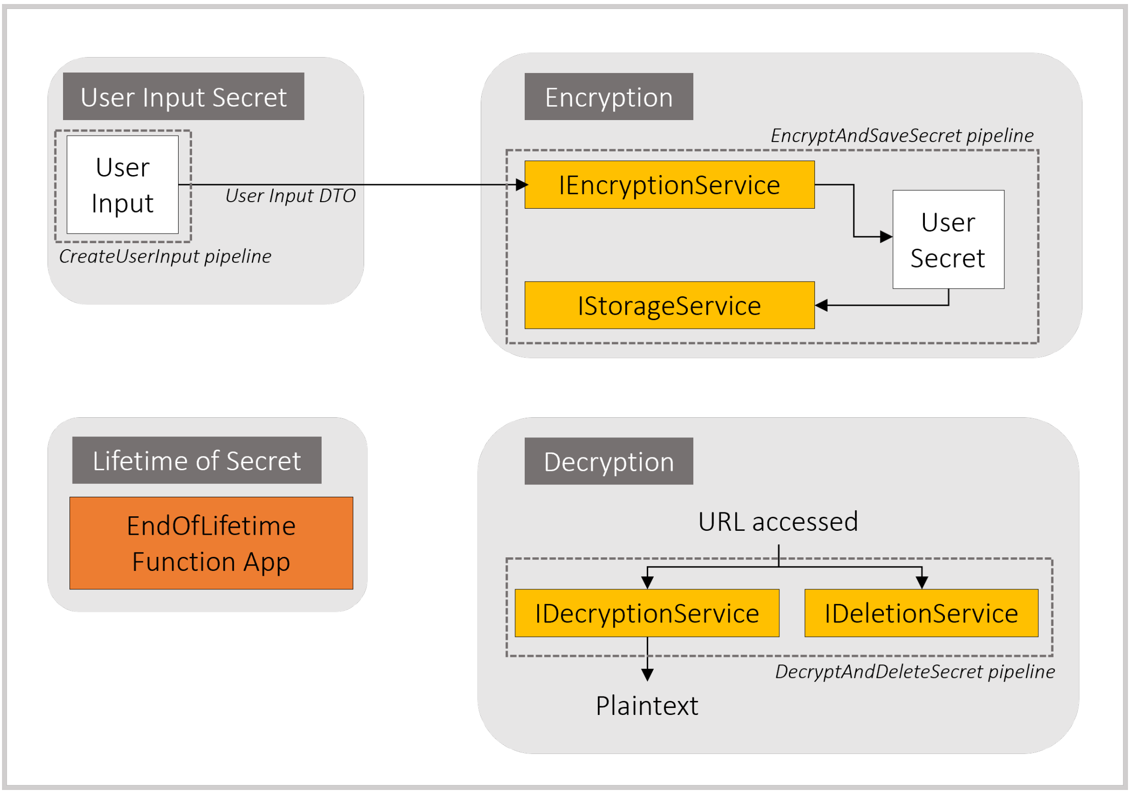


Figure 4.1: The domain model of the Bastion secret sharing solution for anonymous users. The data flow is demonstrated by the arrows between services and aggregate roots. MediatR is used to send requests to pipelines in the domains. The pipelines are shown as dotted squares, the aggregate roots and services as white and orange boxes, respectively.

The domain model contains the following four domains, which are described in detail in the sections below:

- User Input Secret
- Encryption
- Decryption
- Lifetime of Secret

The domain model for the secret sharing solution with added authorization and authentication can be seen in Figure 4.2. It is similar to the domain model in Figure 4.1, but with added steps for authentication prior to creating or accessing a secret.

The user can still choose to remain anonymous, and simply share the generated URL with their intended recipient. Or they can login and authenticate themselves with their Azure AD account, select their intended recipient(s), and copy the URL for sharing. If the secret has intended recipient(s), a user accessing the URL will need to authenticate themselves to be able to view the secret.

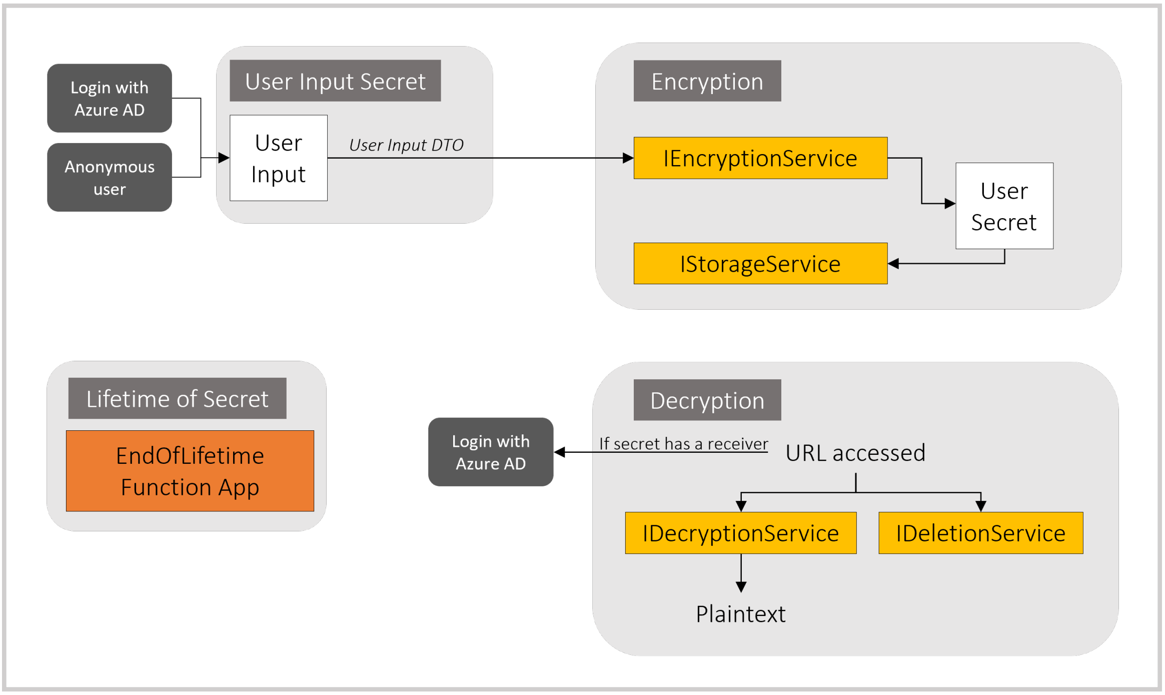


Figure 4.2: The domain model of the complete Bastion secret sharing solution. The data flow is demonstrated by the arrows between services and objects. MediatR is used to send requests to pipelines in the domains. The aggregate roots and services are shown as white and orange boxes, respectively.

The domain model has two aggregate roots, which are shown in Figure 4.3. They have no entities connected to them, only values. The ID, TimeStamp, Lifetime, OIDSender and OIDReceiver in the UserInput aggregate root corresponds to the ID, TimeStamp and Lifetime in the UserSecret aggregate root. The OIDSender has a default value set to an empty string. The OIDReceiver is by default a string array declared as null. If a user has logged in and authenticated themselves prior to creating the secret, they can choose one or more receivers from the drop-down menu. This will populate the OIDSender and OIDReceiver values in both of the aggregate roots. If a user chooses remain anonymous and does not authenticate themselves, the default values of the OID values are utilized.

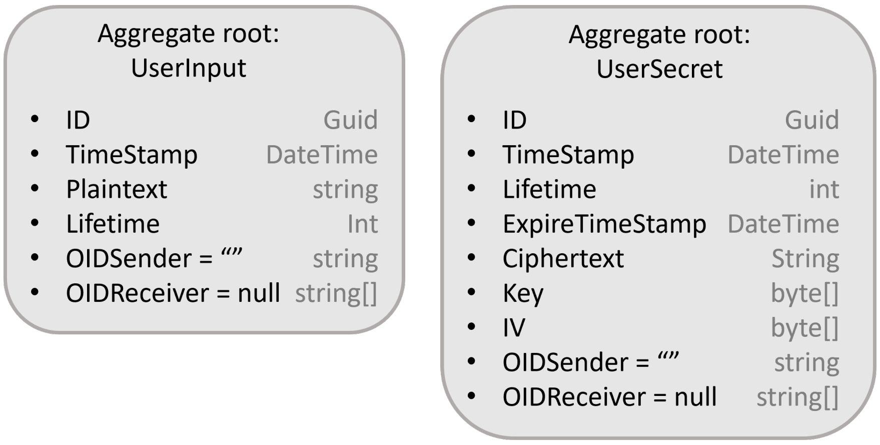


Figure 4.3: The aggregate roots of the Bastion secret sharing solution.

4.3 Setup of domain model in code

The domain model is divided into sections of code representing each of the domains in Figure 4.1. Found in the core domain in the folder structure, each subdomain contains the following:

- **Pipelines (dotted squares)**: Handles incoming requests from the razor pages
- **Services (orange boxes)**: Services are utilized by the pipelines to perform operations on, and with, the request input data
- **Aggregate root (white boxes)**: Located in the domain they belong to
- **Data transfer object (text following arrow)**: Located in the origin domain from where they will be transferred to a different domain

Helpers and managers are placed outside of the core domain, as these are general methods utilized in more than just one domain.

This code setup is chosen to decouple and reuse code, and to make maintenance simple. The services are decoupled from the pipelines, and the pipelines decoupled from the razor pages. The versatile helpers can be utilized anywhere in the code.

The User Input Secret domain has the following build-up:

- **Pipeline**: CreateUserInput, which takes the form input from the user, and creates and returns a UserInput object
- **Services**: None
- **Aggregate root**: UserInput
- **Data transfer object**: UserInputDto, which is utilized to send data from the User Input Secret domain to the Encryption domain

The Encryption domain has one pipeline and two services:

- **Pipeline**: EncryptAndSaveSecret, which receives the UserInputDTO as input, encrypts the secret using the EncryptionService and stores it in blob storage in the correct format using the StorageService.
- **Services**: EncryptionService And StorageService
- **Aggregate root**: UserSecret
- **Data transfer object**: None
- **Helper class**: UserSecretJsonFormat

The Decryption domain has a similar setup as the encryption domain, but has no aggregate root:

- **Pipeline**: DecryptAndDelete secret, which receives an ID and possibly OIDReceiver(s) as input, decrypts the secret if it exists in storage using the DecryptionService, and deletes it from storage using the DeletionService. The secret is returned to the receiver.
- **Services**: DecryptionService and DeletionService
- **Aggregate root**: None
- **Data transfer object**: None

4.4 User Input Secret Domain

The User Input Secret domain consists of one aggregate root, the User Input, and one data transfer object (DTO) class for transferring data from the User Input Secret domain to the Encryption domain.

This domain receives the user's input which consists of a secret message in plaintext and the lifetime of the secret as specified by the anonymous or authenticated user. The domain also receives the OID of the sender and receiver(s) if the user is authenticated. The User Input object is created by a pipeline, and it can be seen in Figure 4.3.

4.4.1 UserInput

Secret ID

The ID, which is be used to identify the secret in storage, and in the secret URL, is generated using the Guid.NewGuid() C# method. It creates a Version 4 Universally Unique Identifier (UUID), which contains 122 bits of strong entropy [25].

TimeStamp

The time stamp is in UTC time at the time of creation of the UserInput object.

Plaintext

The plain text is the secret message which will be encrypted and stored.

Lifetime

The lifetime is a value given in hours, and can either be 1, 8 or 24 hours. It defines how long the secret will be stored, starting from creation time stamp, before it is deleted if no one accesses the secret. The maximum limit of 24 hours is set for security purposes, to avoid a user creating a secret with an insecure length of life.

OIDSender and OIDReceiver

The OID of the receiver(s) and sender are fetched using the Microsoft Graph permissions if the sender has authenticated themselves when accessing the application. If the user does not log in, the default values are used.

The OIDSender and OIDReceiver values in the aggregate roots in Figure 4.3 represents the unique identifier for Azure AD accounts, the OID. This value is chosen to identify users because it is unique and immutable, which names and emails may not always be. Without access to the AD tenant, the OID itself does not reveal any information about a user's identity.

These values could be hashed prior to being stored in blob storage, but they are currently not as it is a wanted feature to be able to display the sender of the secret to the receiver(s). The functionality is present in the solution, and can be used in the future if deemed necessary.

4.4.2 Secret URL

The returned secret URL will be in the following format:

https://bouvet-bastion.azurewebsites.net/<ID>

Where the ID is the same ID as in both aggregate roots in Figure 4.3.

4.5 Encryption domain

The Encryption domain consists of a pipeline, one aggregate root and two services. The domain receives the User Input DTO. The plaintext is encrypted into a ciphertext by the EncryptionService, and a User Secret object is created. The object can be seen in Figure 4.3.

4.5.1 UserSecret

The values ID, TimeStamp, Lifetime, OIDSender and OIDReceiver are equivalent to the values in the UserInput aggregate root.

ExpireTimeStamp

The expire time stamp is set by adding the Lifetime integer hour value to the TimeStamp DateTime object value. The new time stamp represents when a secret expires, in UTC time.

Ciphertext

The ciphertext represents the encrypted message in string format.

Key

The key represents the symmetric decryption key for the ciphertext in a byte array.

IV

The IV value represents the initialization vector for encryption and decryption in a byte array.

4.5.2 AES key generation and encryption

The method Aes.Create() generates a new key and initialization vector (IV) for encryption and decryption, with a default key size of 256 bits. The default value is used. The key must be kept secret, as it is a symmetric key. The initialization vector does not need to be kept secret, but should be. It is renewed for every encryption session and is never reused [24]. The method call of Aes.Create() returns the system default class of of AES implementation in .NET [24].

4.5.3 File naming convention

The files stored in Azure blob storage are named after the following convention

<ID>–<YYYY-MM-DDTHH:MM:SSZ>.json

Where the ID corresponds to the ID of the secret, as seen in Figure 4.3, and the ID in the URL. The timestamp in the file name corresponds to the expiration time and date of the secret. The date-time format is ISO8601, and "Z" indicates the time zone UTC.

4.5.4 JSON format for storing secrets without the key

A separate class, UserSecretJsonFormat, has been created to take input in the form of a User-Secret object and transform it into a data structure in JSON format fit for storage. The class can be seen in Figure 4.4. The class does not contain the symmetric key or the OIDReceiver, but is otherwise identical to the UserSecret aggregate root in Figure 4.3. A helper method creates a UserSecretJsonFormat object, serialises the object, and returns the JSON string data which is then uploaded to blob storage.

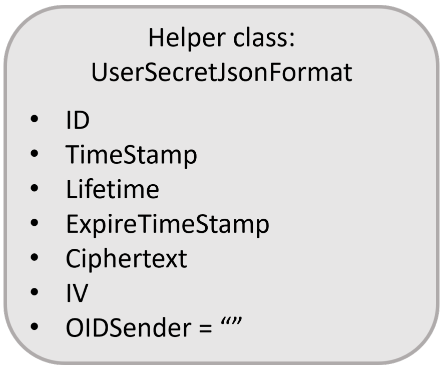


Figure 4.4: The UserSecretJsonFormat class, used as a helper class for storing secrets in blob storage without the symmetric key, which is stored separately in a key vault.

4.5.5 Secret and key storage

After the UserSecret object is created, the StorageService saves the secret to storage in Azure. The symmetric key is stored in a key vault in Azure, with the ID as the key name. The secret, as shown in Figure 4.4, is stored in an Azure blob storage container. The name of the file which can be found in the Azure storage container is as described in Section 4.5.3.

The choice to store the ciphertext and key separately is a security measure decided during the threat modeling prior to implementation. Simply obtaining either the key or the ciphertext separately gives no way to access the plaintext message. To decrypt the secret message, the key, initialization vector and the ciphertext need to be obtained together.

4.6 Decryption domain

The decryption domain consists of a pipeline and two services. Once a secret URL is accessed, the pipeline retrieves the secret from storage, and the DecryptionService decrypts the ciphertext and returns the plaintext message to be displayed. The DeletionService then deletes the key in the key vault and the secret in the blob storage container.

4.6.1 AES decryption

The same key and initialization vector used to encrypt the secret are brought together when the anonymous user or an intended receiver accesses the URL, and the secret gets decrypted. The secret and key gets deleted afterwards, and the key and IV are never reused. The Aes.Create() method is used to create an instance of the AES class, and the key and IV values are set equal to the values used to encrypt the plaintext message in the first place.

4.7 Lifetime of Secret domain

The fourth domain consists of an Azure function application, named EndOfLifeTime. It checks if any secrets are expired, and deletes any that are. The description of the function application is given in Section 4.9.3.

4.8 Authorization and authentication

4.8.1 Authorized and unauthorized view

The web application has components visible depending on whether a user is anonymous or authenticated. If a user is not logged in, the page simply looks like the anonymous sharing solution in Figure 4.9. If the user logs in, they are given the ability to chose one or more receivers for their secret, as seen in Figure 4.14.

The "<AuthorizeView>" and "<Authorized>/<Unauthorized>" handles which components on the index page and display secret page can be seen by the user visiting the web application. This is an all or nothing authorization level, either you are authenticated, or you are not. There are, for example, no administrator pages in the web application. When logging in, the user is redirected to an SSO page where they can chose an Azure AD account. After logging in, they are redirected back to the index page, now with more functionality visible.

4.8.2 Azure AD tenant and application registration

For testing purposes, an Azure AD tenant has been created for the Bastion web application. A few selected users have been added to the tenant. This has been done because the application registration requires administrative consent to be able to access information about all the users registered in the AD tenant.

The Microsoft Graph API permissions given to the Bastion web application are the following:

- User.Read
- User.ReadBasic.All

4.8.3 OAuth 2.0 and OIDC flow

The OAuth 2.0 framework with OpenID Connect protocol handles the authorization and authentication of the Azure AD users registered in the Azure AD tenant. The identity provider is Azure AD, and the flow used to authenticate and authorize user is the code flow shown in Figure 2.4. The authorization is handled by OAuth 2.0, and the authentication by OpenId Connect.

The application uses the library Microsoft.Identity.Web to sign-in and sign-out users from the web application. When signing in, ID Tokens are obtained from the Azure AD tenant. The application is a single tenant application, meaning that only users registered in the Azure AD tenant created for testing can log in and utilize the secret sharing solution as authenticated users. The web application does not accept login with other Azure AD tenants, or other IdPs.

To be able to retrieve data on the user which is currently logged in, and of all other users in the tenant, the web application needs the following details in appsettings.json:

- Domain of Azure AD tentant
- Tenant ID of Azure AD tenant
- Client ID of application registration
- Client secret of the application registration

The reason for wanting to retrieve data from the Azure AD tenant to the web application is to:

- Obtain the OID of the sender of the secret
- Show the sender a list of names of possible receivers
- Obtain the OID(s) of the receiver(s) chosen

4.8.4 Validation of received tokens

Before the the Index and ViewSecret pages are loaded when a user is logged in, the authenticated user's token is validated by confirming the following values:

- Tenant ID: Should be the tenant ID of the application registration of Bastion.
- Issuer: Should be an URI in the format of: "https://login.microsoftonline.com/<tenant ID>/v2.0"

4.8.5 Logic for sharing a secret with a group of predefined receivers

The OIDReceiver is an array of strings, as seen in Figure 4.3. Its default value is an empty string array, in the case of an anonymous user creating a secret. When the authenticated creator of the secret chooses a receiver from the drop-down menu, the individual's OID is added to the OIDReceiver array.

The receivers of the secret are then stored in the metadata of the blob as key-value pairs. The key is the OID stripped of hyphens, containing only numbers and letters. This is due to the naming policy of Azure storage accounts [32]. The value belonging to the key is a string value, by default set to "false", which indicates that the receiver has not yet accessed the secret. This process can be seen in Figure 4.5 below.

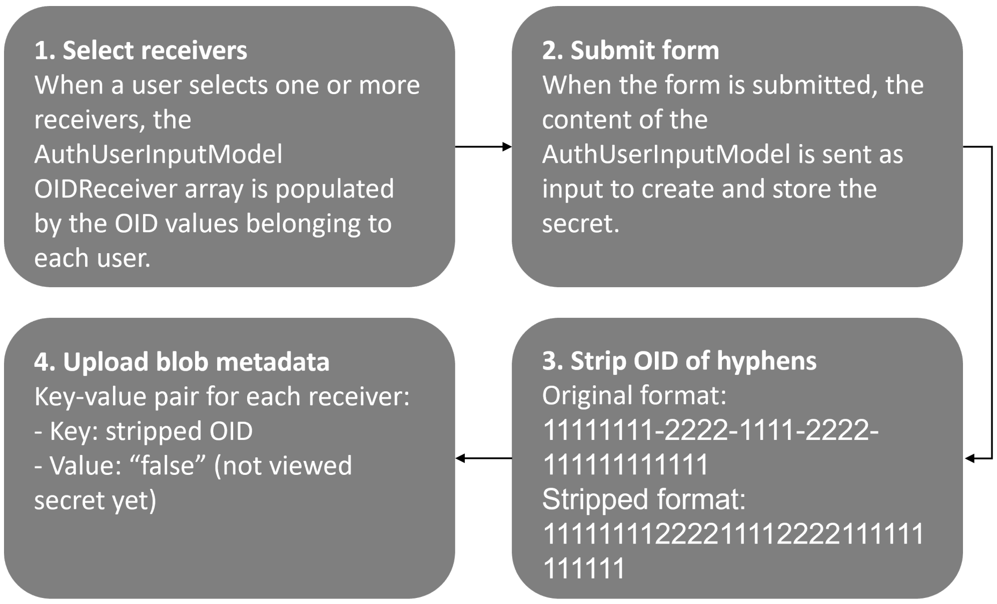

1. Select receivers
When a user selects one or more receivers, the AuthUserInputModel OIDReceiver array is populated by the OID values belonging to each user.
2. Submit form
When the form is submitted, the content of the AuthUserInputModel is sent as input to create and store the secret.
3. Strip OID of hyphens
Original format: 11111111-2222-1111-2222-111111111111
Stripped format: 11111111222211112222111111111111
4. Upload blob metadata
Key-value pair for each receiver:
- Key: stripped OID
- Value: “false” (not viewed secret yet)

Figure 4.5: The process of storing receivers in the metadata belonging to blob containing a secret.

When a secret is accessed through the URL, the OID of the authenticated user is compared with the receivers found in the metadata. If the OID is found as one of the keys in a key-value pair, the user is an intended recipient. If the value is "false", the user has not yet viewed the secret, and is granted access to view it.

The blob’s metadata is then updated to reflect that the user will be accessing the secret. The value in the relevant key-value pair is changed to "true", and the metadata is updated. If the user is not found as an intended recipient in the metadata, they are not allowed to access the secret.

Before a secret is deleted, a validation is done to confirm all intended receivers have viewed the secret. If the count of key-value pairs in the blob metadata is zero, meaning that the secret has no receivers, or if all OID keys have "true" as their value, the secret can be deleted. When all receivers have visited the URL once, all the key-value pairs will have their value set to "true". The process is demonstrated in Figure 4.6.

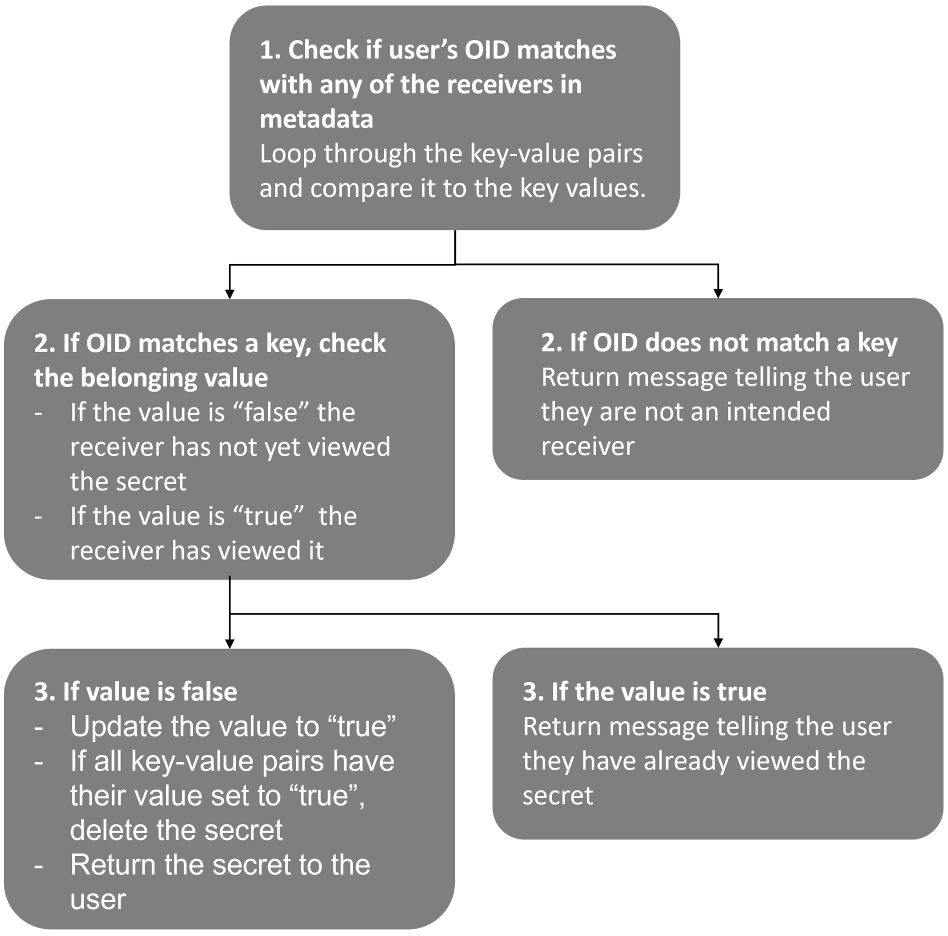

1. Check if user's OID matches with any of the receivers in metadata
Loop through the key-value pairs and compare it to the key values.
2. If OID matches a key, check the belonging value
- If the value is "false" the receiver has not yet viewed the secret
- If the value is "true" the receiver has viewed it
2. If OID does not match a key
Return message telling the user they are not an intended receiver
3. If value is false
- Update the value to "true"
- If all key-value pairs have their value set to "true", delete the secret
- Return the secret to the user
3. If the value is true
Return message telling the user they have already viewed the secret

Figure 4.6: The process of validating receivers in the metadata belonging to a blob containing a secret, prior to letting a user accessing said secret.

4.9 Resources in Azure

The secret sharing solution is made up of the following resources in Azure, which all reside in one resource group:

- Web Application: Bastion
- Function Applcation: EndOfLifetime timer trigger
- Storage account
- Key vault
- User-assigned managed identity

The resources and how they interact with each other can be seen in Figure 4.7. They all interact through the roles given to them by the managed identity instance. The resources are all hosted in Norway. The reason for this is to store the secret messages in an area of common law, but also to reduce latency since most users are expected to be in Norway.

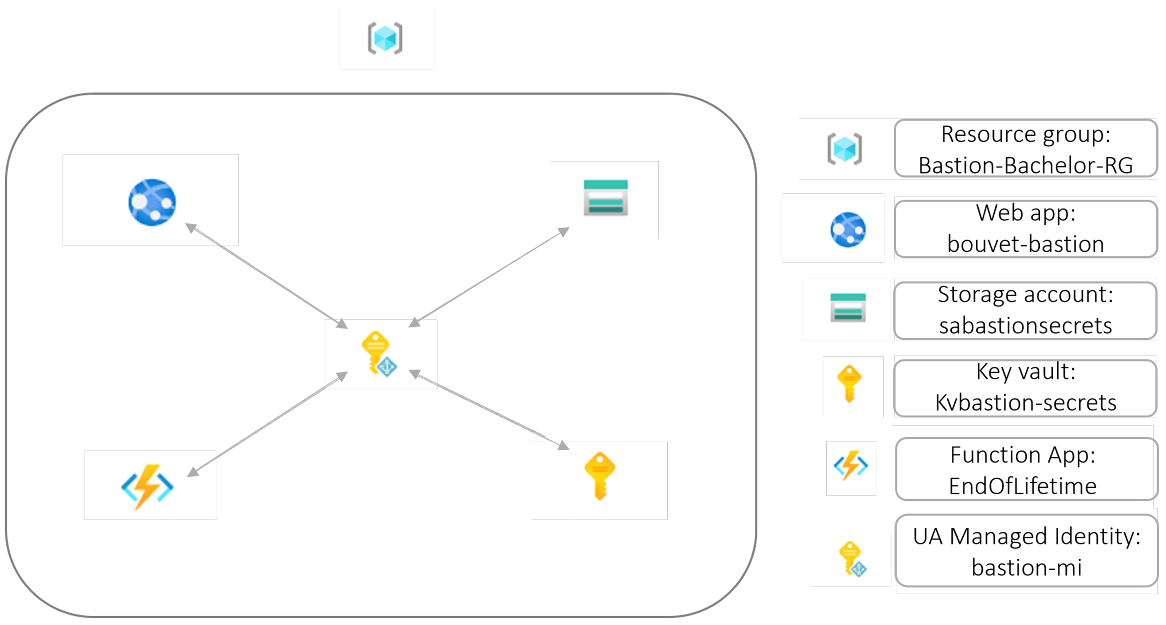


Figure 4.7: The resources in Azure which make up the secret sharing solution.

4.9.1 User-assigned Managed Identity

A user-assigned managed identity is used to enable the function and web application to access and perform actions on items in the storage account and key vault. The motivation for using managed identity is to avoid having to use secret connection strings when interacting with these resources in Azure. This is a more secure option, and offers easier control over permissions.

The managed identity has the following role assignments connected to two resources:

- Storage account, sabastionsecrets: Storage Blob Data Contributor
- Key vault, kvbastion-secrets: Key Vault Contributor

These roles are built-in roles in Azure. Using the principle of least privilege, the applications will be able to get, list, set, delete and purge blob files or secrets, but will not be able to delete or create containers, view or change access control, or any other elevated actions. When the applications are added to the managed identity instance, they get the same access permissions as stated in the role assignments.

The user-assigned managed identity client ID is needed in the code to be able to authenticate using Default Azure Credentials. It is not considered a secret, and is stored as an environment variable. Using the managed identity instance, tokens are acquired using Default Azure Credentials.

4.9.2 Bastion Web Application

The web application contains a Blazor server-side application which holds the functionality for the domains User Input Secret, Encryption and Decryption.The application is deployed to Azure from Visual Studio. Blazor server-side is a framework for building web applications which can be run in ASP.NET Core [20].

The web application can be visited at: `https://bouvet-bastion.azurewebsites.net/`.

Managers

The managers are classes containing methods which are utilized by the Blazor server-side application. They are dependency injected in Program.cs. The web application has two managers:

- Logging Manager: This manager consists of methods for handling logging of events, traces and exceptions which are sent to Application Insights in Azure.
- Copy to Clipboard Manager: This manager copies text to clipboard. Utilized for copying URLs and decrypted secrets.

Helpers

The web application has three helper classes:

- GetUserAssignedDefaultCredentialsHelper: This helper sets up the default credentials with the user assigned managed identity client ID to be able to access the storage account and key vault.
- GetSecretFromKeyVaultHelper: This helper retrieves a secret from a key vault given a secret name as input.
- HashingHelper: This helper creates a hash based on an input value, and verifies a hash value against an input value.

Only the first two are currently used. The HashingHelper has not been utilized for the OID of sender and receiver(s) when storing the secret, but this is something that can be implemented if deemed necessary.

4.9.3 EndOfLifetime Function Application

Functionality

The function application accesses the file names of all the files currently stored in the Azure blob storage container. By parsing the file name, which contains the secret's expiration time stamp, the function application deletes all secrets which are expired, and does nothing if expiration of lifetime is not reached.

The function application converts the time stamp to a DateTime object, and compares it with the current date and time, DateTime.UtcNow(). If the time stamp object is in the past, the function application deletes the secret in the blob storage container, and the key in the key vault.

The function application is deployed from Visual Studio to Azure.

Timer trigger interval

The EndOfLifetime function application is a timer trigger which is triggered once every five minutes to check the status of lifetime of the currently stored secrets.

Managers

The managers are classes containing methods which are utilized by the timer trigger. They are dependency injected in Startup.cs. The function application has two managers:

- Logging Manager: Methods for handling logging of events, traces and exceptions which are sent to Application Insights in Azure.
- Storage Manager: Methods for handling getting lists of file names, checking the time stamps in them, and deleting and purging expired secrets.

Helpers

The function application has two helper classes:

- GetUserAssignedDefaultCredentialsHelper
- GetSecretFromKeyVaultHelper

These are the same helpers as in the Bastion web application in Section 4.9.2.

4.9.4 Storage Account

The storage account has a container which is used for storing the JSON format text files, seen Figure 4.4. The storage account provides geo-redundant storage and secure transfer, and requires TLS version 1.2 as a minimum. Data stored in the storage account are encrypted with Microsoft-managed keys, in addition to infrastructure encryption. Access to the storage account is managed by the managed identity instance as described above.

The storage account has a life cycle rule for deletion of blobs, versions and snapshots in the containers secrets-test and secrets. It deletes all blobs one day after creation. This is in place in addition to the timer trigger function application, and acts as a "safety net" in case the function application should fail or not run.

In addition to this life cycle rule, soft delete for blob storage has also been disabled. This means that when either the web or function application deletes a blob, it is deleted permanently and cannot be restored.

4.9.5 Key vault

The key vault is used to store the symmetric keys and the Application Insights connection string used for logging in both the Bastion web application and the EndOfLifetime function application.

The key vault has an access policy set up that is connected to the user assigned managed identity described in Section 4.9.1. Both the web and function application are given their access rights from the managed identity instance. The managed identity instance has the following access policy set. Using the principle of least privilege, only the necessary access permissions have been assigned to it:

- Get: Get a secret from key vault
- Set: Set a secret in key vault
- List: List all secrets in key vault
- Delete: (Soft) delete a secret from a key vault
- Purge: Permanently delete a secret from a key vault

By default, all Azure key vault only have soft delete enabled for secrets. To completely delete the keys from the key vault, the purge protection has been turned off. After deleting the secret, the soft deleted secret is then purged to be removed completely. After purging, the key cannot be restored.

This operation of deletion and purging is done both by the Bastion web application after a secret has been accessed, and by the EndOfLifetime function application when the lifetime of a secret expires.

4.9.6 Logging: Application Insights

To assure non-repudiation, logging is implemented for both the EndOfLifetime function application, and the Bastion web application. Both of the applications are connected to the same Application Insights in Azure, and the logs are stored in the belonging storage account described above.

The following operations are logged:

- Anonymous secret creation
- Authenticated secret creation, secret creator and receiver (OIDs)
- Secret encryption
- Secret stored in key vault and storage container
- Secret is accessed anonymously

- Secret is accessed by an authenticated user (OIDs)
- If the wrong user has tried to access a secret that was not meant for them (OIDs)
- Secret decryption
- Secret is deleted and purged
- Every function call of EndOfLifetime
- If EndOfLifetime has deleted any secrets
- If none of the secrets are expired, and EndOfLifetime does not delete any secrets

4.10 Pages and models of the Bastion web application

4.10.1 Razor pages

The Bastion web application contains the following pages:

- Index: Front page where messages can be encrypted and the generated URL copied to share the secret.
- DisplaySecret: Page to display a secret when accessing a valid URL, or an error message in case the secret does not exist, or because the user is not an intended recipient.
- About: Description of the web application
- Logout: Page redirecting to the index page following logout
- Oops: User is directed to this page in case they try to access an invalid secret URL format
- Error: The page displayed in case an error occurs.

4.10.2 Page models

The Index page has two models:

- UserInputModel: Model for anonymous user input
- AuthUserInputModel: Model for authenticated user input

As seen in Figure 4.8, the models set the default input values, the maximum string length is set to 5000 characters, and the range of lifetime from 1 to 24 hours. All fields are required.

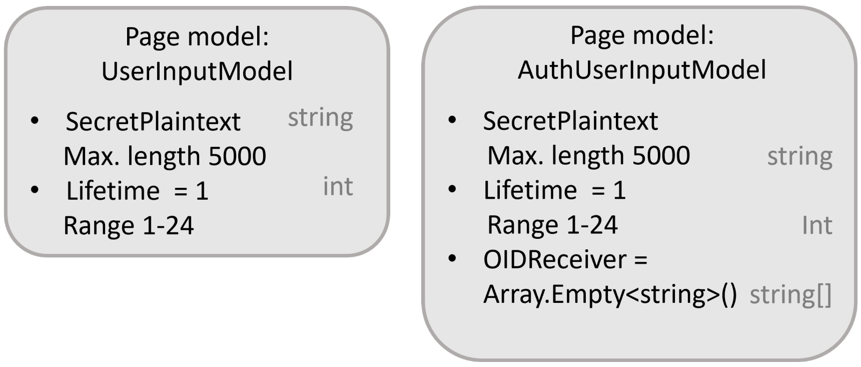


Figure 4.8: The page models of the Bastion secret sharing solution.

4.11 Design of the Bastion web application

The application has been made mobile-friendly, and the colors utilized are standard Bouvet colors.

4.11.1 Creating and accessing a secret as an anonymous user

The index page of the Bastion web application is seen in Figure 4.9. In this page the user can type in or paste a secret, choose the lifetime, and copy the one-click link once it has been generated, as seen in Figure 4.10.

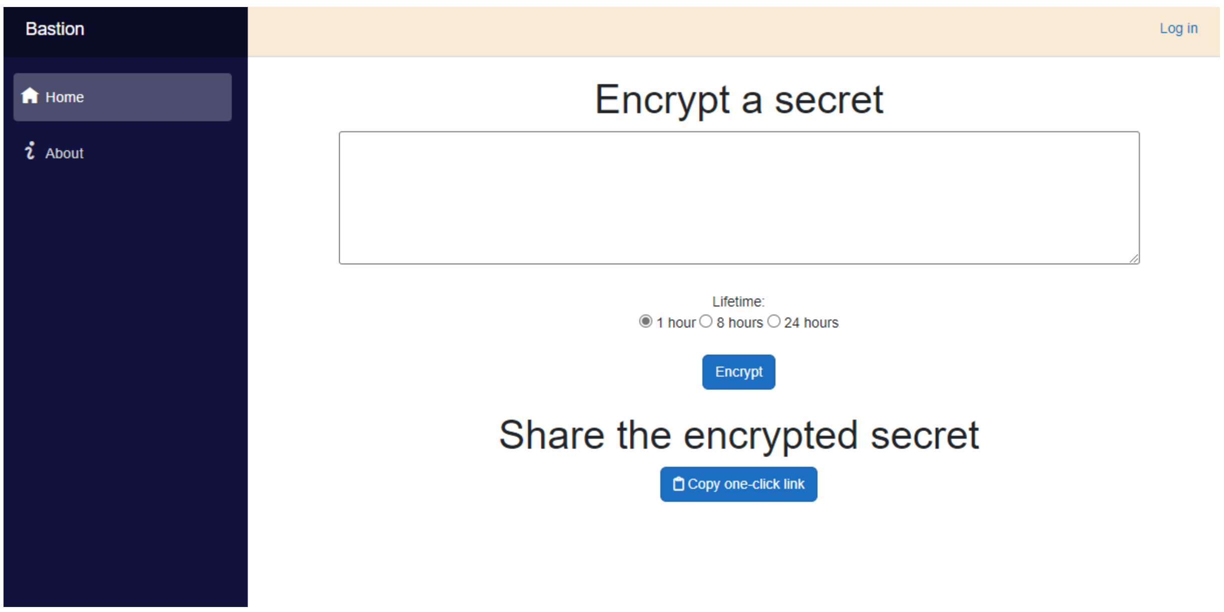
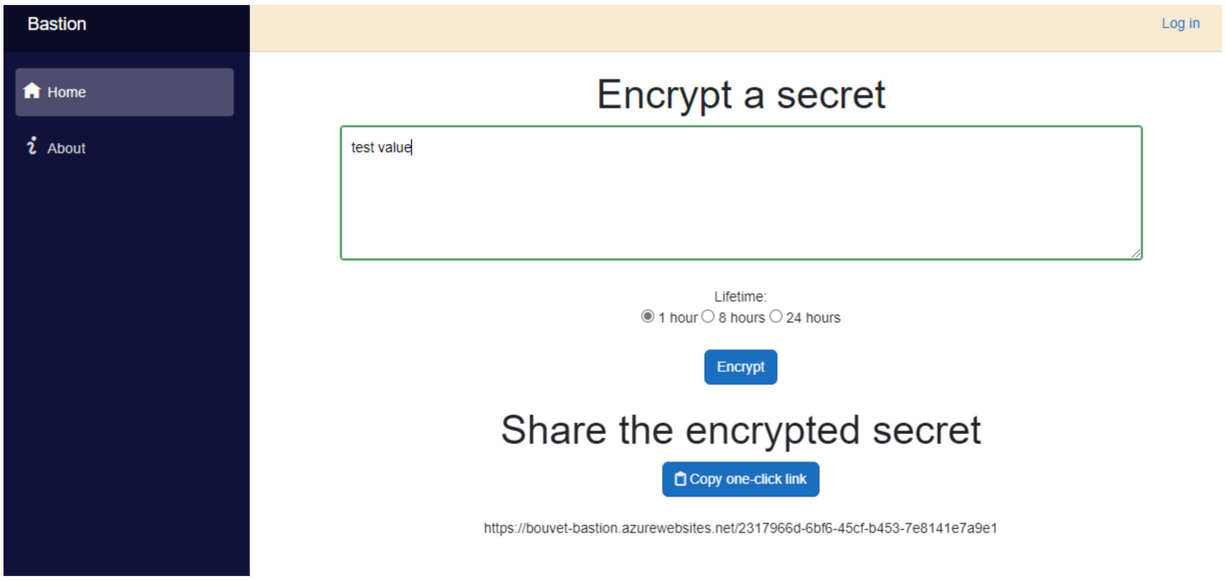


Figure 4.9: The front page of the Bastion web application for an anonymous user.



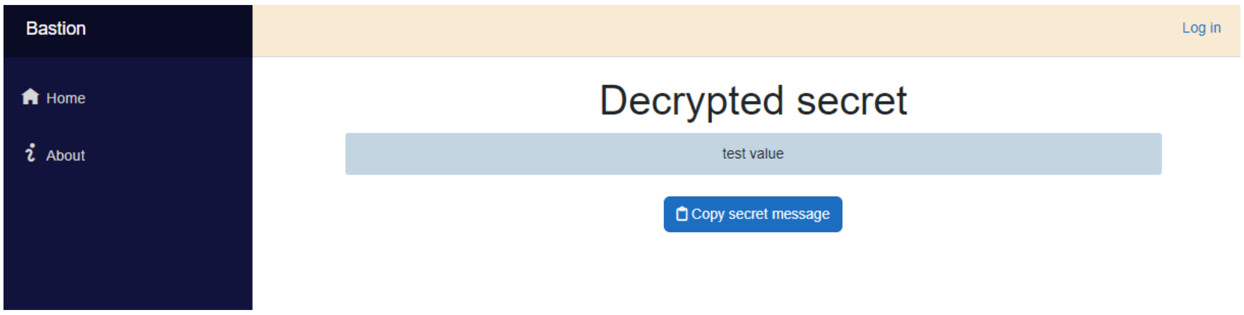
test value

Lifetime: [✓ 1 hour] 8 hours 24 hours

https://bouvet-bastion.azurewebsites.net/2317966d-6bf6-45cf-b453-7e8141e7a9e1

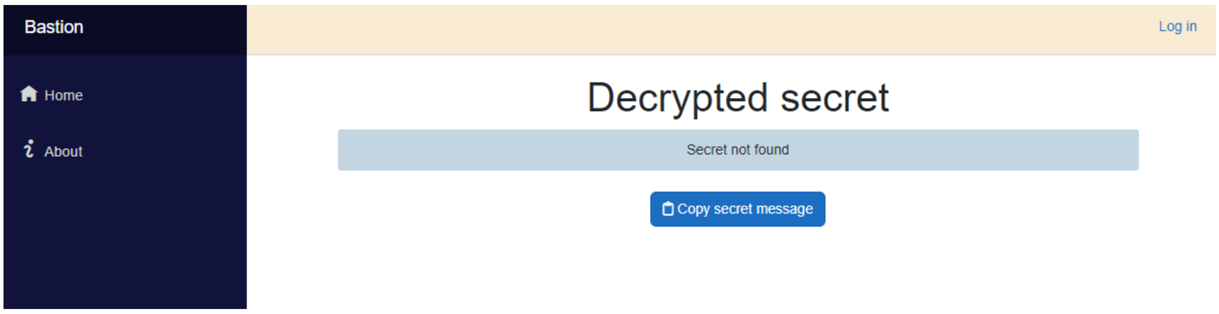
Figure 4.10: The front page of the Bastion web application for an anonymous user after a one-click link has been generated and the secret encrypted and stored.

After the one-click link is shared with a receiver, the user will be able to read the message in plaintext when accessing the URL, as seen in Figure 4.11. If the secret does not exist, or if it has already been accessed (and consequently deleted), the user will get the following message seen in Figure 4.12. The anonymous user does not need to enter a key to access the secret, only access the URL.



test value

Figure 4.11: The view secret page of the Bastion web application for an anonymous user when the URL is accessed.



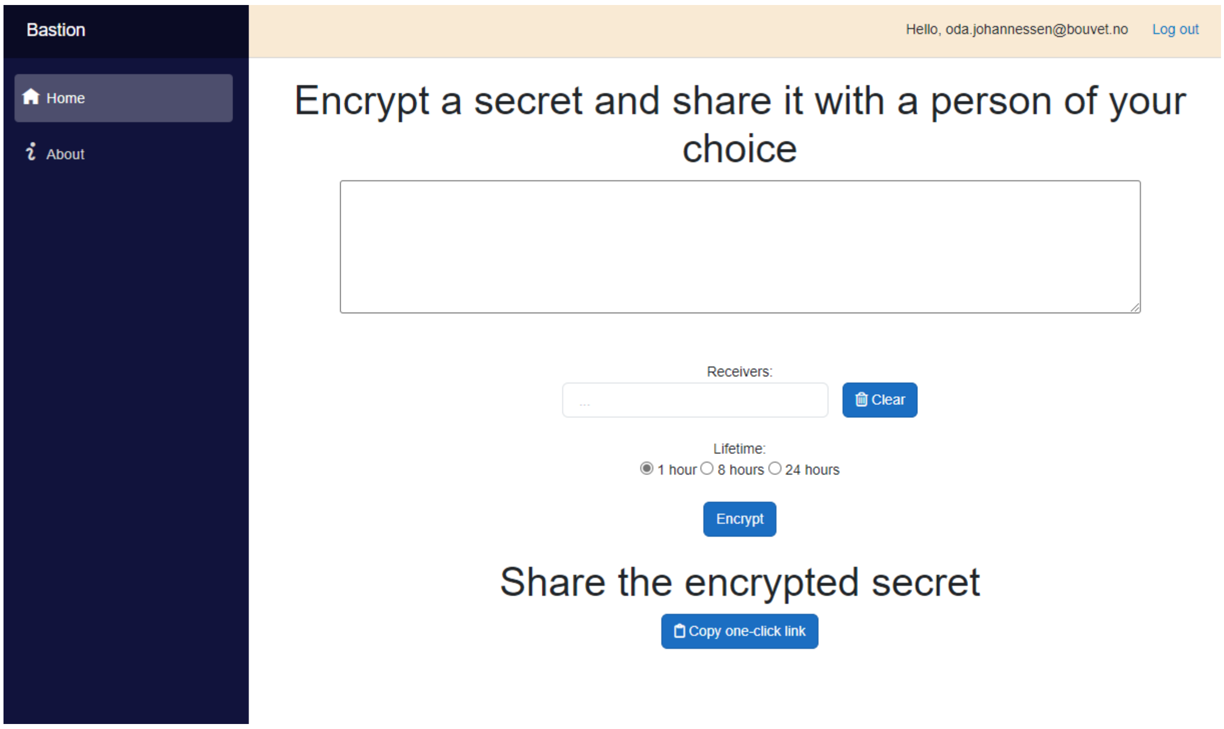
Decrypted secret

Secret not found

Figure 4.12: The view secret page of the Bastion web application for an anonymous user when the URL is accessed and the secret does not exist.

4.11.2 Creating and accessing a secret as an authenticated user

When selecting "Log in" in the upper right corner, the user is redirected to Microsoft's login page where they can choose a user for single-sign on. After a successful login, the user is redirected to the index page in Figure 4.13. The email of the current user is shown in the upper-right corner, along with the "Log out" button. The chosen receiver of the secret can be seen in the receivers input pane in Figure 4.15.



Hello, oda.johannessen@bouvet.no

Encrypt a secret and share it with a person of your choice

Receivers:

Lifetime:
[✓ 1 hour] 8 hours 24 hours

Share the encrypted secret

Figure 4.13: The front page of the Bastion web application for an authenticated user. No receivers have been chosen.

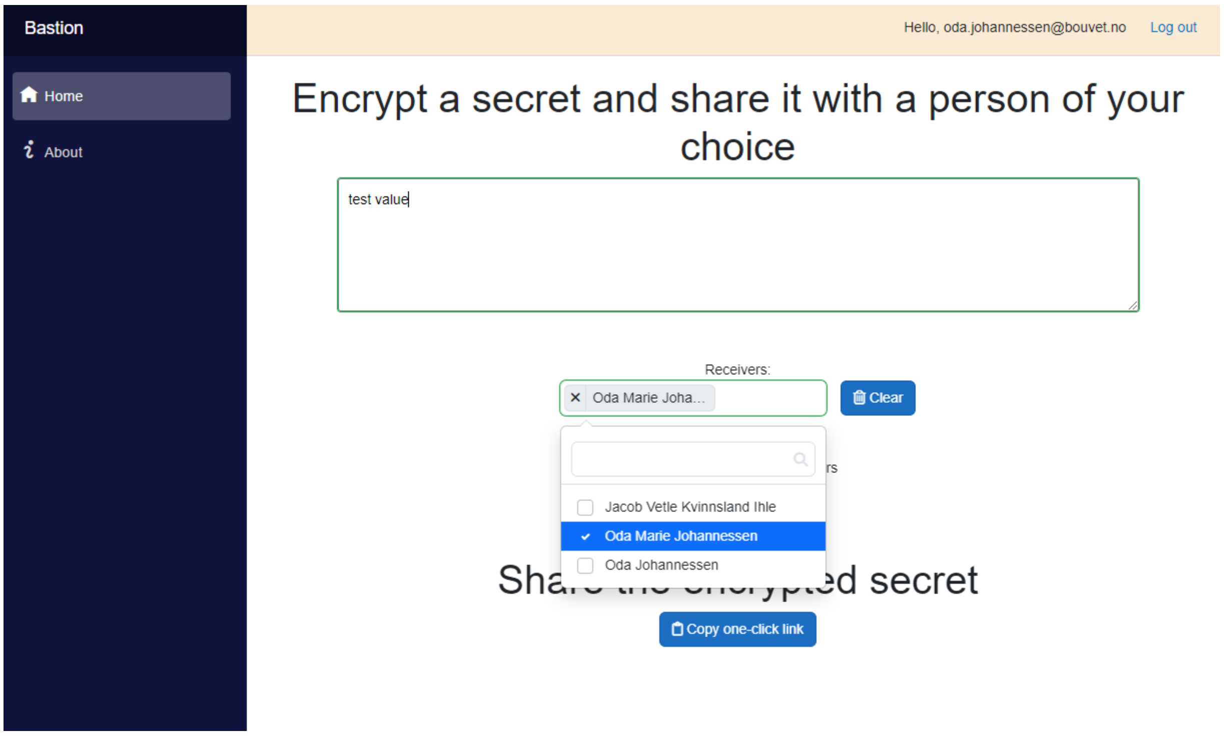
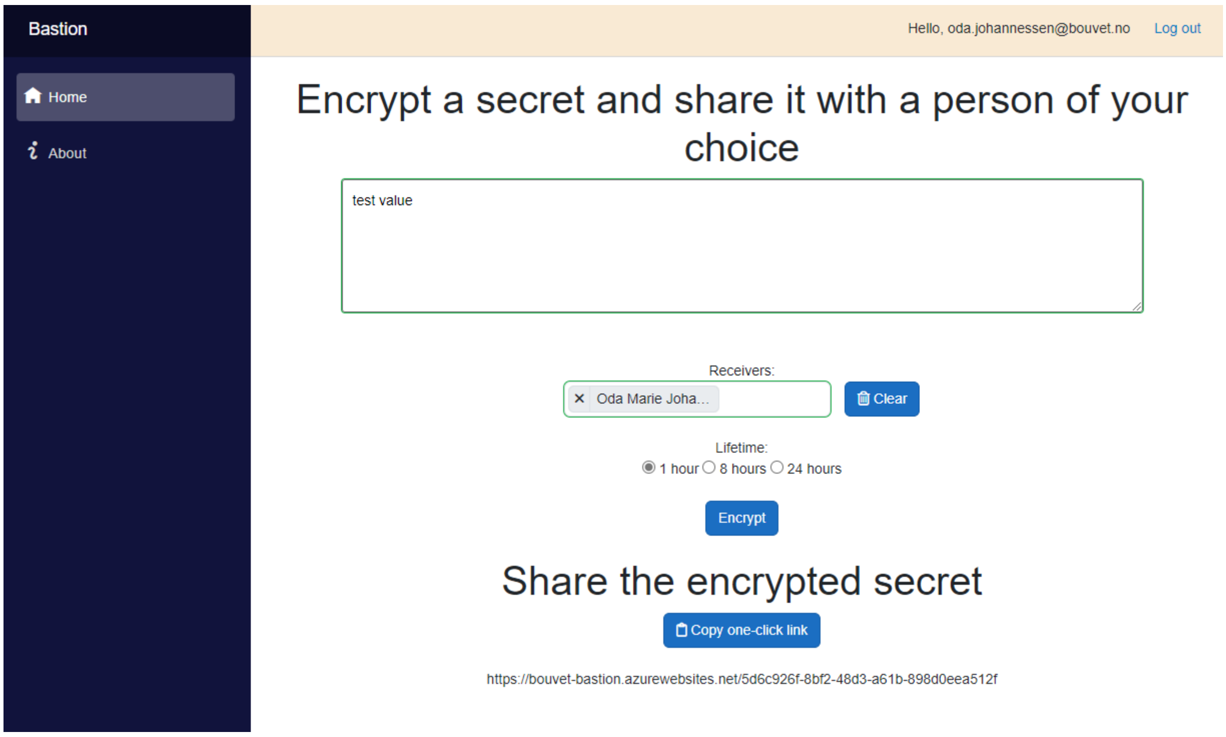

Bastion
Hello, oda.johannessen@bouvet.no
Log out
Home
About
Encrypt a secret and share it with a person of your choice
test value
Receivers:
Oda Marie Joha...
Clear
Jacob Vetle Kvinnsland Ihle
Oda Marie Johannessen
Oda Johannessen
Share the encrypted secret
Copy one-click link

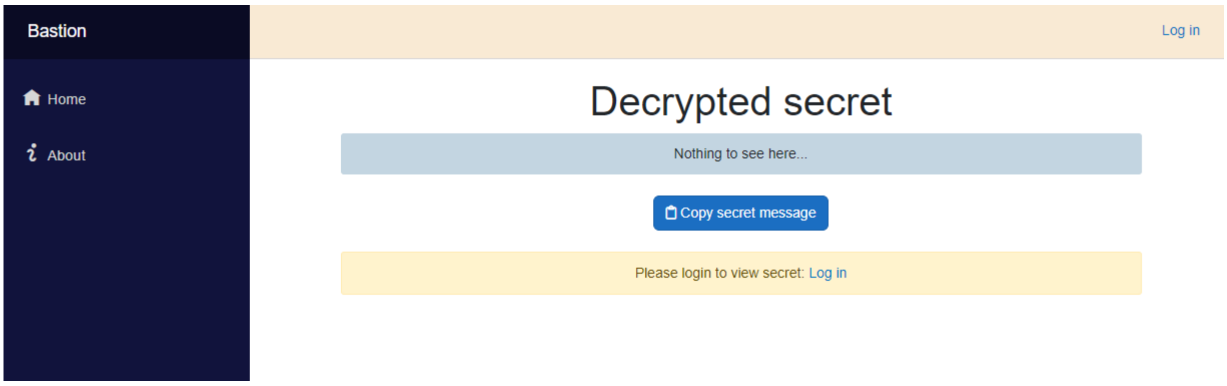
Figure 4.14: The front page of the Bastion web application for an authenticated user when they are choosing receivers. The drop-down menu has a search function. The "clear" button removes all receivers currently selected.


Bastion
Hello, oda.johannessen@bouvet.no
Log out
Home
About
Encrypt a secret and share it with a person of your choice
test value
Receivers:
Oda Marie Joha...
Clear
Lifetime:
[✓ 1 hour] 8 hours 24 hours
Encrypt
Share the encrypted secret
Copy one-click link

https://bouvet-bastion.azurewebsites.net/5d6c926f-8bf2-48d3-a61b-898d0eea512f

Figure 4.15: The front page of the Bastion web application for an authenticated user after a one-click link has been generated and the secret encrypted. One receiver has been chosen in this example.

If the secret the user is trying to access has predefined receivers, but the user is currently not logged in, the user is prompted to login as shown in Figure 4.16.



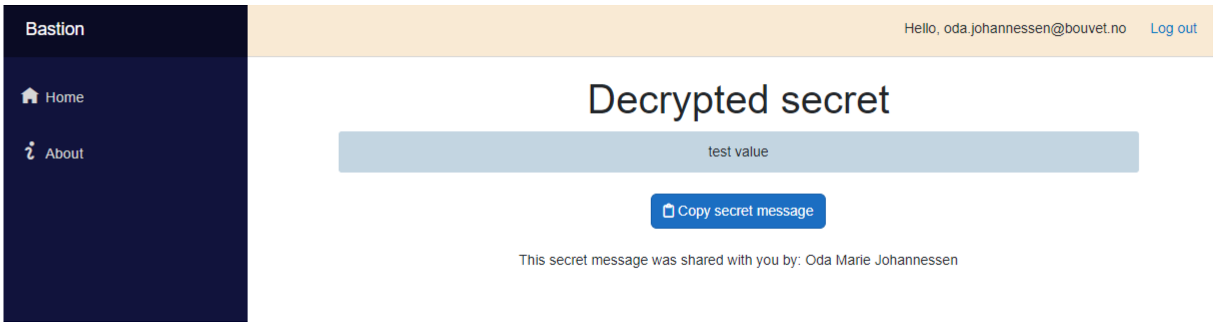
Decrypted secret

Nothing to see here...

Please login to view secret: Log in

Figure 4.16: The view secret page of the Bastion web application for an anonymous user where the secret has predefined receivers, and the user is prompted to log in.

If the user logs in, and they are the one of the intended recipients, the secret will be shown, along with the sender of the secret, as shown in Figure 4.17. If the user is not an intended receiver, or has already viewed the secret once, the following message is shown as in Figure 4.18.



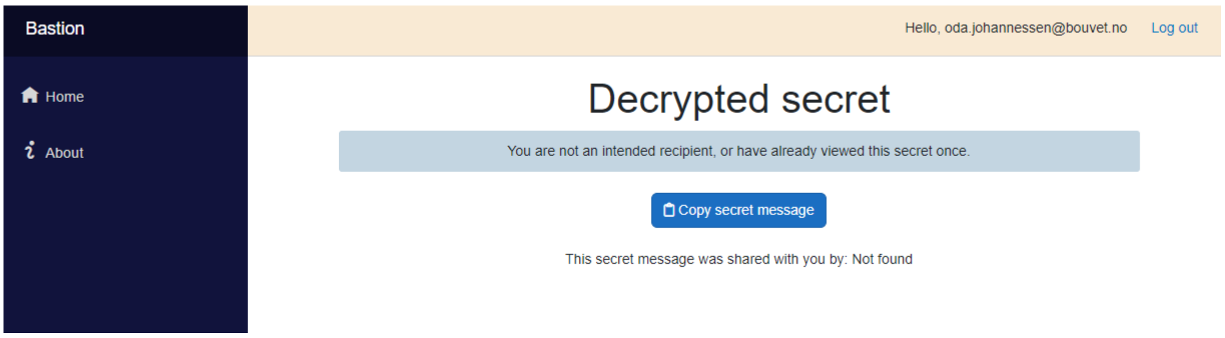
Hello, oda.johannessen@bouvet.no

Decrypted secret

test value

This secret message was shared with you by: Oda Marie Johannessen

Figure 4.17: The view secret page of the Bastion web application for an authenticated user who is an intended receiver.



Hello, oda.johannessen@bouvet.no

Decrypted secret

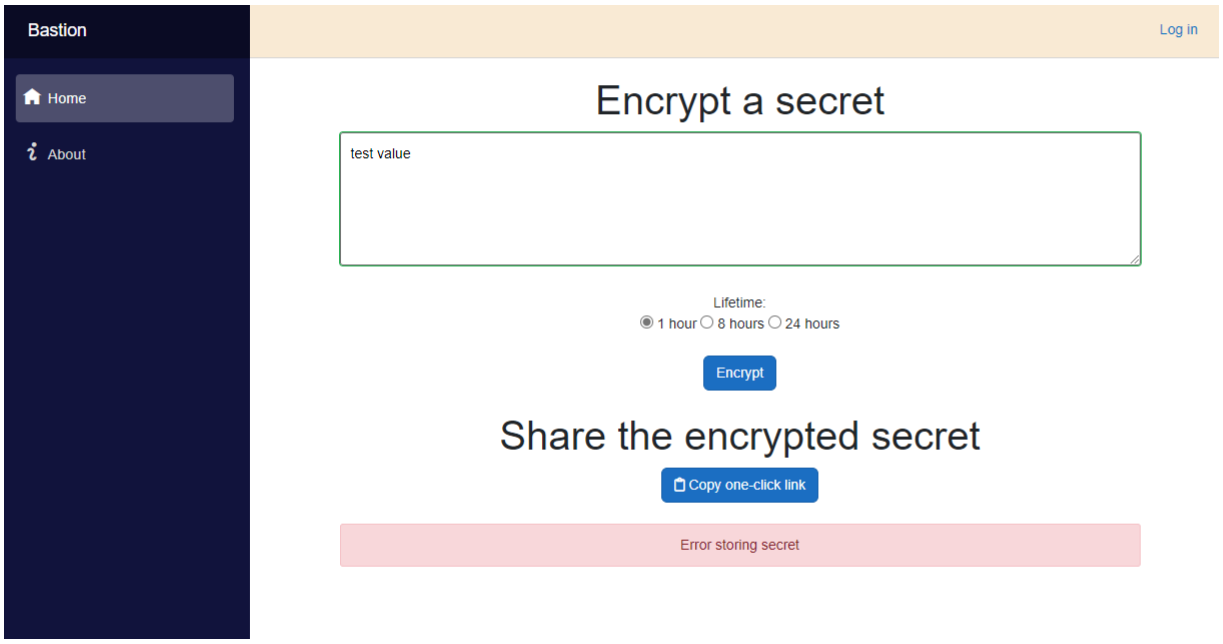
You are not an intended recipient, or have already viewed this secret once.

This secret message was shared with you by: Not found

Figure 4.18: The view secret page of the Bastion web application for an authenticated user who is not an intended receiver, or who has already viewed the secret once.

4.11.3 Error messages in case of unsuccessful storing of secret

In case of issues with storing the encrypted secret, an error message will appear instead of a secret one-click link, as shown in Figure 4.19.



Encrypt a secret

test value

Lifetime:
● 1 hour ○ 8 hours ○ 24 hours

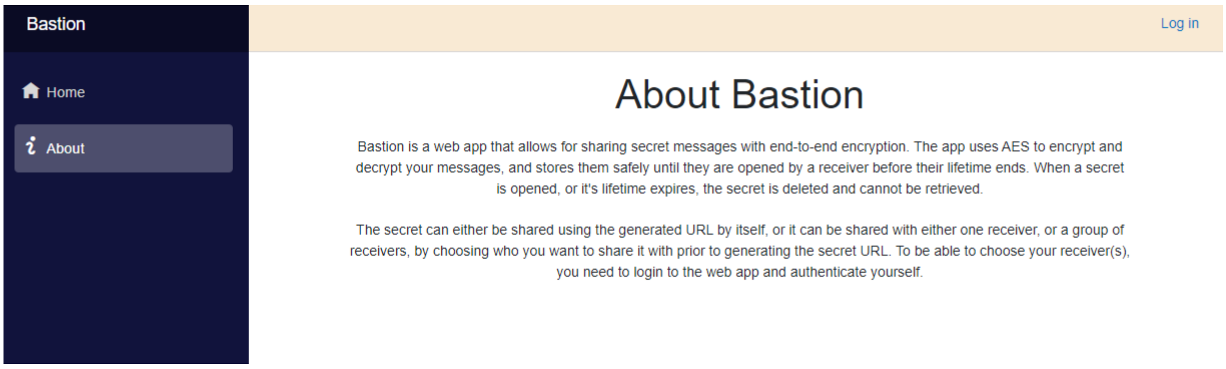
Share the encrypted secret

Error storing secret

Figure 4.19: The front page of the Bastion web application in case of an error occurring during storage or encryption of the secret.

4.11.4 About page

The about page describing the sharing solution can be seen in Figure 4.20.



About Bastion

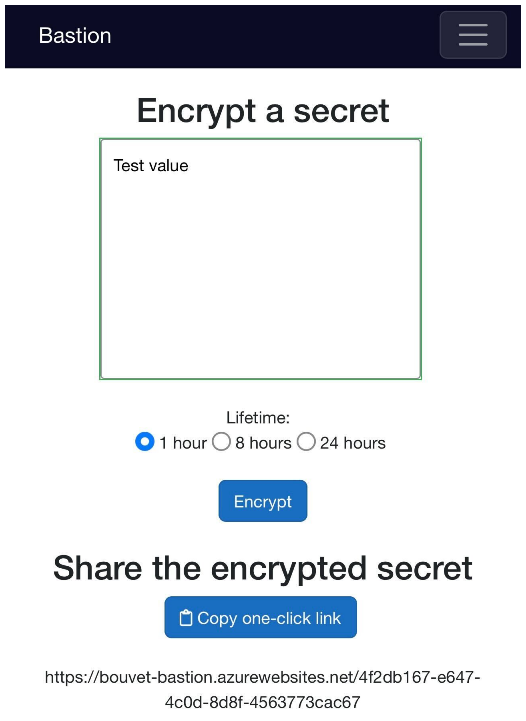
Bastion is a web app that allows for sharing secret messages with end-to-end encryption. The app uses AES to encrypt and decrypt your messages, and stores them safely until they are opened by a receiver before their lifetime ends. When a secret is opened, or it's lifetime expires, the secret is deleted and cannot be retrieved.

The secret can either be shared using the generated URL by itself, or it can be shared with either one receiver, or a group of receivers, by choosing who you want to share it with prior to generating the secret URL. To be able to choose your receiver(s), you need to login to the web app and authenticate yourself.

Figure 4.20: The about page of the Bastion web application, describing the secret sharing solution.

4.11.5 Mobile application version

The mobile version for anonymous users can be seen in Figure 4.21.


Bastion

Encrypt a secret

Test value

Lifetime:
[✓ 1 hour] 8 hours 24 hours

Encrypt

Share the encrypted secret

Copy one-click link

https://bouvet-bastion.azurewebsites.net/4f2db167-e647-4c0d-8d8f-4563773cac67

Figure 4.21: The layout of the front page of the Bastion web application for an anonymous user in mobile format.

4.12 GitHub workflow

A GitHub workflow has been created for the repository. The workflow builds the project and runs all unit tests for every commit or pull request to the main branch.

4.13 Unit testing

Three unit tests have been created for the Bastion web application solution. They can be found in the Bastion.Tests Xunit project.

- UserSecret test: Creates a UserSecret object, and confirms that the input values are in accordance with the resulting object.
- UserInput test: Creates a UserInput object, and confirms that the input values are in accordance with the resulting object.

- Encryption decryption round trip test: Encrypts and then decrypts a message, and compares the decrypted message with the input text.

Chapter 5

Solution Evaluation

5.1 Implementation of the threat modeling results

The threat modeling was not originally part of the scope of the thesis, but was added to the plan at the commencement of the thesis work. The reason for including threat modeling was because it is relevant for the security focus of this thesis.

The outcome of the threat modeling were the mitigation strategies that were followed throughout the development phase of the solution. These specific mitigation techniques were very helpful to have during development. The use cases and data flow diagrams where also beneficial, because they showed a clear picture of what the core functionality of the secret sharing solution was going to be. This helped to further develop the the domain model and compose the basis for the code structure.

In addition to the mitigation strategies, the exercise of doing a threat modeling itself added a perspective of security to all the consequent work, which in itself was very helpful.

5.2 Utilizing domain-driven design concepts

The utilization of domain-driven design proved helpful in defining the functionality of the solution. The resulting web and function application code is decoupled into domains and easy to maintain. The logic in the services are completely separated from the pipelines which utilizes them, and the domains in the domain model are represented clearly in the code structure.

5.3 Architecture of storage of data

The storage solution which was chosen, combining a storage account for blob storage of the secret object, and the key vault for the secret keys, proved to work well.

Separating the storage of the encrypted secret and its details from the key was one of the mitigation strategies decided during threat modeling. Even if a threat actor managed to obtain access to a secret in the storage account, the encrypted message on its own is useless without the encryption key stored in the more secure location of the key vault.

The choice to store the symmetric keys in the key vault could have been replaced by sharing the key with the user who created the secret, making the user responsible for sending the key to their recipient along with the one-click link. This would have simplified the storage solution by removing the need for a key vault.

This approach was not chosen, because the solution should be as simple as possible for the user, and because Yopass already has this functionality available. Taking predefined receivers into account, a URL falling into the wrong hands would not be an issue, since only the intended receivers would be able to access the secret. Thus storing the key separately from the secret works as intended.

The current infrastructure architecture of Bastion utilizing blob storage supports file sizes up to 8 TiB [31]. This is convenient in case of an expansion of the solution. The current solution limits the size of the plaintext input to 5000 characters, but has potential to scale easily.

5.4 Choice of cryptographic algorithm

The AES algorithm used is the current recommended algorithm for block cipher encryption by NIST [41]. Triple DES could also have been utilized, but was not because it will be disallowed after 2023, when the transition period for replacing it by AES ends [2]. AES is also faster and safer than Triple DES, and is considered safe against brute-force attacks for all three key lengths [2].

5.5 Replacing the current ID generation scheme

The GUID class currently used for generation of secret IDs should not be used for cryptographic purposes [25], and needs to be replaced by a generator meant for cryptographic objectives. Microsoft's RandomNumberGenerator class is recommended for this case, and is a viable alternative that the GUID class can be replaced by [25]. This change was not done during the thesis work due to time constraints.

5.6 Potential security issues and risks

A current potential security risk is that the client secret of the application registration is stored in the appsettings.js file of the application. A more secure option would be to store it in the key vault, and reference it to the web applications configuration, and from there retrieve it to the appsettings.js file. This was not done due to lack of time during the thesis work.

5.7 Reoccurring issue for the BlobClient upload method

The BlobClient in the library Azure.Storage.Blobs has several built-in methods, among them the Upload method. Utilizing this method to upload secret content to a blob in blob storage failed inconsistently when the web application was running in Azure, and this error was not possible to reproduce during local testing. The error message was as follows:

Error uploading secret to blob storage: 'Server failed to authenticate the request. Make sure the value of Authorization header is formed correctly including the signature.

The solution to this issue was to write a HTTP put request to upload content to a blob from the ground up, and include the correct version of the header "x-ms-version" in line 19 below:

```
 1 // Upload to blob
 2 BlobClient client = new BlobClient(new Uri(uriSA), credentials);
 3
 4 // Workaround for the reoccurring header issue for the web app
 5 TokenRequestContext requestContext = new TokenRequestContext(new[] {
       "https://storage.azure.com/.default" });
 6 AccessToken token = await credentials.GetTokenAsync(requestContext);
 7 string accessToken = token.Token;
 8
 9 byte[] secretByteArray = Encoding.UTF8.GetBytes(secretJsonFormat);
10 using (MemoryStream ms = new MemoryStream(secretByteArray))
11 {
12     //var uploadResponse = await client.UploadAsync(ms);
13     HttpRequestMessage request = new HttpRequestMessage()
14     {
15         Method = HttpMethod.Put,
16         RequestUri = new Uri(uriSA),
17         Content = new StreamContent(ms)
18     };
19     request.Headers.Add("x-ms-version", "2020-04-08"); // Solves the
           issue
20     request.Headers.Add("x-ms-blob-type", "BlockBlob");
21     request.Headers.Add("Authorization", "bearer " + accessToken);
22
23     HttpClient httpClient = new HttpClient();
```

```
    HttpResponseMessage response = await
       httpClient.SendAsync(request);

    if (!response.IsSuccessStatusCode)
    {
        logging.LogException($"Error uploading secret to blob. ID:
           '{userSecret.Id}'. ");
        return false;
    }
```

5.8 Possible purging race condition

During the process of (soft) deletion and purging (hard deletion) of keys in the key vault for the EndOfLifetime function application and for the Bastion web application, there occasionally arises an error during the purging process. For the EndOFLifetime timer trigger this happens if the interval of the timer trigger is too small, approximately less than five minutes. It can also happen if there is not a short delay set up between the deletion process and the purging process, as it does in the web application.

This appears to be a race condition, where deletion and purging actions are performed on the same key at once. This disturbs the purging process, but not the deletion process.

The consequence is that if the interval is shorter than five minutes, a key can possibly only be soft deleted, and not permanently deleted. However, the encrypted message is always hard deleted from blob storage, so even if the key itself is restored, there is no message to decrypt it with.

To mitigate this issue the run interval of the EndOfLifetime is set to five minutes, and a delay between the deletion and purging process is added in both the function and web application, but a more sophisticated solution could be to utilize locks to mitigate the issue.

5.9 Testing

The secret sharing solution should have a higher code coverage than it currently has. Due to lack of time, creating more tests was not made a priority. The solution has three unit tests, but this should be expanded to include both more unit tests and integration tests.

5.10 Expanding to other identity providers

The step to expand to other identity providers than Azure AD, such as AWS or Google, was not complete due to time constraints. It was also not prioritized due to both Bouvet and a large amount of Bouvet's customers primarily using Azure AD as their identity provider, and not any of the other well-known identity providers.

Chapter 6

Discussion

6.1 Lifetime of secrets

One of the most important security requirements for the sharing solution is the guaranteed deletion of secrets after they are opened or they expire. No secret or key should be stored for longer than 24 hours in the Azure storage account or key vault.

When a secret has been opened, either by all receivers, or as an anonymously shared secret, it is deleted from storage by the web application. The deletion of secrets which are not opened is handled by a timer trigger function application which is run once every five minutes. Both the web and function application handles deletion by both soft deleting and purging the secret data, meaning that neither key nor encrypted message can be restored.

In addition to the function application, the storage account blob container has a backup rule set for how long a blob is allowed to be stored prior to being deleted. The maximum amount of time a blob is allowed to exist in the container is 24 hours after creation.

The key vault does not have a similar backup rule, because the key vault also stores other secret values utilized by the web and function applications. However, the key itself will be useless without the encrypted message in case the function application fails and the blob container rule goes into effect, and lifetime is still respected.

6.2 Sharing a secret with no predefined receivers

When sharing a secret which has no predefined receivers, either as an anonymous or authenticated user, it is important to note that if the one-click link falls into the wrong hands, anyone can view the secret. The link should not be forwarded or shared with anyone but the intended recipient of the secret message. The recipient, or reader of the secret, will always be anonymous in the case of a secret shared with no predefined receivers.

Sharing a secret using the anonymous option will still be preferable to simply sending a secret in plaintext, because the secret has a limited lifetime, and will either be deleted when it expires or after it has been opened.

6.3 Storing receivers in metadata compared to in the blob text file

The initial plan was to store the receivers of a secret in the JSON format class in Figure 4.4 as an array of strings. This string array would be found in the blob text file for its respective secret in plaintext.

The decision was made to instead utilize the blob metadata to set receivers of the secret. The main reason for this is to avoid having to download the content of the blob prior to checking if the current user is an intended receiver.

By utilizing blob metadata, the only thing needed to access the receivers is a blob client. The metadata can be downloaded through a get-operation, and viewed, without having to touch the contents of the blob itself. The blob contents will first be downloaded locally if the current user is one of the intended receivers. If not, the web application will return an error if the user is not a receiver, or prompt the user to login in case they have not yet authenticated themselves.

This way, the blob content is only downloaded in case it should be shared with a receiver, which is more effective than downloading it as a default to check for receivers, and also more secure.

6.4 Administrative privileges

A disadvantage to hosting the solution as a set of resources in Azure, is that administrators with access to the storage account and key vault, will in theory be able to access and decrypt secrets of other users. The naming convention in storage is to utilize the GUID ID as the name of both the blob file and the key. Thus an administrator will be able to find secrets in the storage account and match them with the appropriate key, and decrypt them if they want to.

A way to avoid this could be to hash the IDs with different salts to avoid this being possible in the first place. Persons with access to the storage account and key vault should also only be limited to the persons in control of the sharing solution, and the principle of least privilege should be enforced.

6.5 Azure AD tenant, OAuth 2.0 and OpenID Connect for authorization and authentication

By outsourcing the handling of authorization and authentication to an identity provider such as Azure AD, the solution does not have to take into account having to store digital identities or to perform the authentication process of users.

This simplifies what the solution needs to provide with respect to storage, and it is an advantage that it does not have to store sensitive user data. Azure AD takes care of this by storing the digital identities of the users.

Roles for the web application are defined in the application registration belonging to the Bastion in the Azure AD tenant, and are also kept separate from the web application.

Azure AD utilizes OAuth 2.0 and Open ID Connect protocols to complete the authentication and authorization of users. The user is thus redirected from the web application to log in, and redirected back after a successful log in-attempt. By utilizing robust industry standard protocols for authentication and authorization, common access control and identification and authentication vulnerabilities, as described in OWASP Top ten [50], could be avoided.

On the other hand, if a self-hosted IdP solution was chosen, Bastion would have to handle storage of the user information. The user would also have to register with the web application, instead of utilizing their already existing digital identity in the Azure AD tenant. Utilizing Azure AD as the identity provider for the web application provides a more secure and user-friendly solution.

6.6 Updated recommendations from NIST in 2024

Last year, NIST announced the first cryptographic algorithms which are designed to be able to withstand attacks from quantum computers. NIST's post-quantum cryptographic standard, which these algorithms will become a part of, is expected to be published in 2024 [46]. Following this new standard, the AES encryption technique used in this thesis might need to be replaced by one of the new quantum-resistant algorithms.

Chapter 7

Conclusion

7.1 What remains before Bouvet can start utilizing Bastion for sharing secrets?

Bastion needs to be expanded from the proof of concept in the current Azure AD testing tenant, and given an application registration in Bouvet's Azure AD tenant, along with the roles as described in this thesis. Bastion can also be made available to customers by making the application multi-tenant, or by letting the application become open source so that they can host it themselves. The current ID generation scheme must be replaced by a random generator which is supported for cryptographic use.

The web application also needs to go through testing and quality assurance by a team in Bouvet, and have its unit and integration testing project expanded to improve the code coverage. A penetration test would also be beneficial to control that there are not any vulnerabilities present in the application or its infrastructure.

The resources in Azure which completes the sharing solution should also have their setup controlled to verify that they comply with Bouvet's security standards.

7.2 What went well?

The primary goal of the thesis work was to create a secret sharing solution which allowed for sharing secrets with predefined receivers using a one-click link. This objective has been achieved.

The threat modeling was successful, and the initial plan for infrastructure in Azure proved to work as expected. The planned handling of expired secrets also went according to plan, and the combination of a function application handling deletion, and the backup blob lifetime rule of maximum 24 hours works well. The domain model created during and after threat modeling also provided good basis for the development process, and there were very few issues which presented themselves.

7.3 Possible expansions and improvements for the future

Expand Bastion to include file uploads

To further make the solution more competitive compared to Yopass or Keeper, which are currently used by Bouvet, it can be expanded to allow for file uploads. The current infrastructure of the sharing solution will be able to support large files since blob storage has been utilized as the storage solution.

Azure AD roles replacing OID for identifying receivers

The current solution of storing the OID of individual receivers in key-value pairs in blob metadata can be replaced by a more sophisticated way of controlling access to secrets. Dynamic roles and groups can be set up in the Azure AD tenant to administer access.

Expand unit tests and add integration testing

The sharing solution has some simple unit tests, but should have a higher code coverage. The testing project can be expanded to include more unit tests, and also integration tests. One integration test of particular interest could be a test case for authorization. If the any breaking changes to the authorization are made, the integration test will fail, and alert of these issues.

CI/CD pipeline for deploying code to Azure resources

The current solution of deploying code directly from Visual Studio to the Azure resources could be replaced by an automatic CI/CD pipeline or a GitHub actions setup.

Bibliography

[1] Auth0. *Passwordless Authentication with SMS*. 2023. URL: https://auth0.com/docs/authenticate/passwordless/authentication-methods/sms-otp (visited on 04/09/2023).

[2] Cobb, M. *What is Triple DES and why is it being disallowed?* 2023. URL: https://www.techtarget.com/searchsecurity/tip/Expert-advice-Encryption-101-Triple-DES-explained (visited on 04/26/2023).

[3] Conklin, L. *D1: Define Security Requirements*. 2023. URL: https://owasp.org/www-project-proactive-controls/v3/en/c1-security-requirements (visited on 01/21/2023).

[4] Conklin, L. *Threat Modeling Process*. 2023. URL: https://owasp.org/www-community/Threat_Modeling_Process (visited on 01/11/2023).

[5] Culafi, A. *LastPass faces mounting criticism over recent breach*. 2023. URL: https://www.techtarget.com/searchsecurity/news/252529329/LastPass-faces-mounting-criticism-over-recent-breach (visited on 04/01/2023).

[6] Daemen, J. and Rijmen, V. "AES proposal: Rijndael". In: *The Rijndael Block Cipher* (1999).

[7] Drake, V. *Threat Modeling*. 2023. URL: https://owasp.org/www-community/Threat_Modeling (visited on 01/11/2023).

[8] Duijm, N. J. "Recommendations on the use and design of risk matrices". In: *Safety Science* 76 (2015), pp. 21–31.

[9] Goldreich, O. *Foundations of Cryptography*. First edition. New York: Cambridge University Press, 2009.

[10] Goldwasser, S. and Bellare, M. *Lecture Notes on Cryptography*. June 2008. URL: https://cseweb.ucsd.edu//~Mihir/papers/gb.pdf.

[11] Haals, J. *Yopass - Share Secrets Securely*. 2023. URL: https://github.com/jhaals/yopass (visited on 05/05/2023).

[12] Hammer-Lahav, E. *Introduction*. 2007. URL: https://oauth.net/about/introduction// (visited on 02/24/2023).

[13] Hatzivasilis, G. "Password-Hashing Status". In: *Cryptography* 1.2 (2017).

[14] Jaatun, M. G. *Domenedrevet design for sikkerhet*. 2022. URL: https://infosec.sintef.no/informasjonssikkerhet/2022/09/domenedrevet-design-for-sikkerhet/ (visited on 01/21/2023).

[15] Keeper Security. *Protect your organization against cyberthreats with zero-trust Enterprise Password Management (EPM)*. 2023. URL: https://www.keepersecurity.com/enterprise.html (visited on 05/05/2023).

[16] Kessler, G. C. *An Overview Of Cryptography*. 2023. URL: https://www.garykessler.net/library/crypto.html.

[17] Keycloak. *Open Source Identity and Access Management*. 2023. URL: https://www.keycloak.org/ (visited on 04/09/2023).

[18] Shar, L. K, Pastore, F. Briand, L. C. Mai, P. X. Goknil, A. and Shaame, S. "Modeling Security and Privacy Requirements: a Use Case-Driven Approach". In: *Information and Software Technology* 100 (Aug. 2018), pp. 165–182.

[19] Microsoft. *Application and service principal objects in Azure Active Directory*. 2023. URL: https://learn.microsoft.com/en-us/azure/active-directory/develop/app-objects-and-service-principals (visited on 04/10/2023).

[20] Microsoft. *ASP.NET Core Blazor hosting models*. 2023. URL: https://learn.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-6.0 (visited on 05/05/2023).

[21] Microsoft. *Authentication vs. authorization*. 2023. URL: https://learn.microsoft.com/en-us/azure/active-directory/develop/authentication-vs-authorization (visited on 03/11/2023).

[22] Microsoft. *Azure Active Directory integrations with authentication protocols*. 2023. URL: https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/auth-sync-overview (visited on 02/24/2023).

[23] Microsoft. *Azure Key Vault basic concepts*. 2022. URL: https://learn.microsoft.com/en-us/azure/key-vault/general/basic-concepts (visited on 04/09/2023).

[24] Microsoft. *Generate keys for encryption and decryption*. 2023. URL: https://learn.microsoft.com/en-us/dotnet/standard/security/generating-keys-for-encryption-and-decryption (visited on 02/18/2023).

[25] Microsoft. *Guid.NewGuid Method*. 2023. URL: https://learn.microsoft.com/en-us/dotnet/api/system.guid.newguid?view=net-6.0 (visited on 02/18/2023).

[26] Microsoft. *Identity Providers for External Identities*. 2023. URL: https://learn.microsoft.com/en-us/azure/active-directory/external-identities/identity-providers (visited on 02/24/2023).

[27] Microsoft. *Introduction to Azure Blob Storage*. 2023. URL: https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction (visited on 03/18/2023).

[28] Microsoft. *OAuth 2.0 authentication with Azure Active Directory*. 2023. URL: https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/auth-oauth2 (visited on 02/24/2023).

[29] Microsoft. *OAuth 2.0 Authorization Framework*. 2023. URL: https://auth0.com/docs/authenticate/protocols/oauth (visited on 03/11/2023).

[30] Microsoft. *Quickstart: Set up a tenant*. 2023. URL: https://learn.microsoft.com/en-us/azure/active-directory/develop/quickstart-create-new-tenant (visited on 04/09/2023).

[31] Microsoft. *Scalability and performance targets for Blob storage*. 2023. URL: https://learn.microsoft.com/en-us/azure/storage/blobs/scalability-targets (visited on 04/04/2023).

[32] Microsoft. *Storage account overview*. 2023. URL: https://learn.microsoft.com/en-us/azure/storage/common/storage-account-overview (visited on 04/09/2023).

[33] Microsoft. *Tenancy in Azure Active Directory*. 2023. URL: https://learn.microsoft.com/en-us/azure/active-directory/develop/single-and-multi-tenant-apps (visited on 04/09/2023).

[34] Microsoft. *User profile attributes*. 2023. URL: https://learn.microsoft.com/en-us/azure/active-directory-b2c/user-profile-attributes (visited on 04/09/2023).

[35] Microsoft. *What is Azure Active Directory?* 2023. URL: https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-whatis (visited on 02/20/2023).

[36] Microsoft. *What is Azure DDoS Protection?* 2023. URL: https://learn.microsoft.com/en-us/azure/ddos-protection/ddos-protection-overview (visited on 03/24/2023).

[37] Microsoft. *What is Azure role-based access control (Azure RBAC)?* 2022. URL: https://learn.microsoft.com/en-us/azure/role-based-access-control/overview (visited on 04/18/2023).

[38] Microsoft. *What is Azure?* 2023. URL: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure/ (visited on 04/10/2023).

[39] Mishra, M. R. and Kar, J. “A Study on Diffie-Hellman Key Exchange Protocols”. In: *International Journal of Pure and Applied Mathematics* 114 (2017), pp. 179–189.

[40] Naor, M. and Yung, M. “Universal One-Way Hash Functions and Their Cryptographic Applications”. In: STOC ’89. New York, NY, USA: Association for Computing Machinery, 1989, pp. 33–43.

[41] NIST. *Block Cipher Techniques*. 2023. URL: https://csrc.nist.gov/Projects/block-cipher-techniques (visited on 01/05/2023).

[42] NIST. *Cryptography*. 2023. URL: https://www.nist.gov/cryptography (visited on 01/05/2023).

[43] NIST. *Glossary*. 2022. URL: https://csrc.nist.gov/glossary/ (visited on 02/24/2023).

[44] NIST. *Hash Functions*. 2022. URL: https://csrc.nist.gov/projects/hash-functions (visited on 03/11/2023).

[45] NIST. *Identity Provider (IdP)*. 2023. URL: https://csrc.nist.gov/glossary/term/identity_provider (visited on 03/12/2023).

[46] NIST. *NIST Announces First Four Quantum-Resistant Cryptographic Algorithms*. 2022. URL: https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms (visited on 04/19/2023).

[47] OAuth Inc. *Authorization Code Flow*. 2023. URL: https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow (visited on 03/11/2023).

[48] OAuth Inc. *Identity Glossary*. 2023. URL: https://auth0.com/docs/glossary (visited on 03/12/2023).

[49] OAuth Inc. *OpenId Connect Protocol*. 2023. URL: https://auth0.com/docs/authenticate/protocols/openid-connect-protocol (visited on 03/12/2023).

[50] OWASP. *OWASP Top Ten*. 2022. URL: https://owasp.org/www-project-top-ten/ (visited on 02/24/2023).

[51] OWASP. *Who is the OWASP Foundation?* 2023. URL: https://owasp.org (visited on 04/09/2023).

[52] Parecki, A. *OAuth 2.0*. 2023. URL: https://oauth.net/2/ (visited on 02/24/2023).

[53] Prakash, A. and Kumar, U. “Authentication protocols and techniques: a survey”. In: *International Journal of Computer Sciences and Engineering* 6.6 (2018), pp. 1014–1020.

[54] Samonas, S. and Coss, D. “The CIA Strikes Back: Redefining Confidentiality, Integrity and Availablity in Security”. In: *Journal of Information System Security* 10.3 (2014).

[55] Uithol, M. *Security in domain-driven design*. Apr. 2008. URL: http://essay.utwente.nl/58268/ (visited on 01/10/2023).

[56] Vernon, V. *Implementing Domain-Driven Design*. Upper Saddle River, NJ: Addison-Wesley, 2013.

[57] Roby, N. Yaga, D. Mell, P. and Scarfone, K. “Blockchain Technology Overview”. In: *NISTIR 8203* (Mar. 2018).

Appendices

Appendix A

Architecture Design Record (ADR)

A.1 Choice of threat modeling process: OWASP and STRIDE mnemonic

- Date: 06.01.2023
- Status: Accepted
- Context: To make sure good decisions are made regarding the security of the web application, a threat modeling process will be carried out to identify threats and mitigation strategies.
- Decision: Use the OWASP threat modeling process as basis for the threat modeling.
- Consequences: Mitigate threats and increase the security of the application.
- Stakeholders: Bouvet
- Other possible solutions: Any other relevant framework for threat modeling.

A.2 Choice of security requirements

- Date: 06.01.2023
- Status: Accepted
- Context: Set security requirements to know what to focus on during threat modeling and development.
- Decision: The security requirements defined in Section 3.1.
- Consequences: Increasing the security of the application by raising awareness of security threats.
- Stakeholders: Bouvet
- Other possible solutions: Any other set of relevant security requirements.

A.3 Choice of developing the solution with Microsoft Azure Cloud Services

- Date: 06.01.2023
- Status: Accepted
- Context: Bouvet and many of Bouvet's customers in Rogaland primarily use Azure Cloud Services over other cloud service providers.
- Decision: The choice to develop the solution to utilize the Azure platform and store secrets in Azure.
- Consequences: Create a solution with functionality which can be easily be expanded further by Bouvet.
- Stakeholders: Bouvet
- Other possible solutions: Other cloud service providers or hosting services.

A.4 Choice of identity provider: Azure Active Directory (AD)

- Date: 06.01.2023
- Status: Accepted
- Context: Need to choose an identity provider to authenticate users against.
- Decision: Using Azure AD, as Bouvet has an Azure AD tenant where all employees have an Azure AD identity.
- Consequences: No need for the web application, or resources in Azure, to maintain or handle user information or secrets.
- Stakeholders: Bouvet
- Other possible solutions: Store and maintain user login information for the web application, use a self-hosted identity provider, or any other third-party external identity provider available.

A.5 Choice of where to geographically host resources: Norway

- Date: 06.01.2023
- Status: Accepted
- Context: Need to decide where to geographically host all resources of the sharing solution.
- Decision: East-Norway.
- Consequences: Lower latency.

- Stakeholders: Bouvet
- Other possible solutions: Any other geographical area Azure provides storage in which is deemed appropriate.

A.6 Choice of programming language: C#

- Date: 06.01.2023
- Status: Accepted
- Context: Azure (Microsoft) is the main cloud platform used by Bouvet.
- Decision: C# is the chosen programming language.
- Consequences: Creating a solution in a programming language used by many in Bouvet.
- Stakeholders: Bouvet
- Other possible solutions: Any other relevant programming language.

A.7 Choice of software framework: .NET

- Date: 06.01.2023
- Status: Accepted
- Context: Azure (Microsoft) is the main cloud platform used by Bouvet.
- Decision: Utilize the .NET framework for development
- Consequences: Creating a web application in a framework which is common in Bouvet.
- Stakeholders: Bouvet
- Other possible solutions: Any other relevant framework.

A.8 Choice of .NET version: 6

- Date: 06.01.2023
- Status: Accepted
- Context: Need to decide which version of .NET framework to use.
- Decision: Utilize .NET 6.
- Consequences: .NET 6 has long term support, which is why it was chosen.
- Stakeholders: Bouvet
- Other possible solutions: The newer .NET 7 (does not have long term support) or older versions.

A.9 Choice of web application framework: Blazor Server-Side

- Date: 06.01.2023
- Status: Accepted
- Context: The framework chosen for the Bastion web application.
- Decision: Blazor Server-side
- Consequences: Can be deployed directly to Azure. Can be combined easily with Azure AD and OAuth2, which are planned to be implemented at a later stage.
- Stakeholders: Bouvet
- Other possible solutions: Blazor WebAssembly or other frameworks.

A.10 Choice of MediatR for sending requests and receiving responses in the Bastion web application

- Date: 06.01.2023
- Status: Accepted
- Context: Blazor can utilize MediatR to send requests to pipelines. Enables communication between domains.
- Decision: MediatR is chosen.
- Consequences: Decoupled code and reduced dependencies.
- Stakeholders: Bouvet
- Other possible solutions: Any other form of event handlers or suitable library.

A.11 Choice of framework for unit testing: XUnit

- Date: 06.01.2023
- Status: Accepted
- Context: A test framework is needed to set up tests for the web application.
- Decision: XUnit
- Consequences: XUnit is a widely used testing framework, and is a good choice as most C# developers will be familiar with it.
- Stakeholders: Bouvet
- Other possible solutions: Other testing frameworks like MSTest or NUnit.

A.12 Choice of cryptographic algorithm: AES

- Date: 06.01.2023
- Status: Accepted
- Context: To securely store secrets, an approved algorithm needs to be implemented to handle encryption and decryption.
- Decision: AES has been chosen because it the standard algorithm recommended by NIST [41].
- Consequences: Choosing an algorithm which is approved and safe to use (as of today) is important to guarantee the protection of the data.
- Stakeholders: Bouvet
- Other possible solutions: Could have chosen TripleDES, which is also an algorithm approved by NIST, but AES is considered superior, and this is why it was chosen over TripleDES [41].

A.13 Choice of storage of secrets: Storage account and key vault in Azure

- Date: 06.01.2023
- Status: Accepted
- Context: The choice was previously made to develop the solution to utilize Azure Cloud functionality.
- Decision: The decision made is to utilize an Azure blob container to store the secrets in, and to store the symmetric keys in an Azure key vault.
- Consequences: Separating the storage of the secrets messages and keys to increase security.
- Stakeholders: Bouvet
- Other possible solutions: Could have also stored the data in an any other type of database.

A.14 Choice of user assigned managed identity to handle role assignment to storage account and key vault

- Date: 11.01.2023
- Status: Accepted
- Context: Need to define how the secret sharing web application should be able to interact with other Azure resources.

- Decision: A user assigned managed identity, with two roles, storage blob data and key vault contributor, set for the storage and key vault, respectively.
- Consequences: By using managed identity, the use of secret connection strings are avoided, strengthening security.
- Stakeholders: Bouvet
- Other possible solutions: Utilize connection strings.

A.15 Choice of basic functionality for the secret sharing solution

- Date: 06.01.2023
- Status: Accepted
- Context: Choice of basic functionality for the secret sharing solution.
- Decision: The primary functionality will be a one-click link with life time of secret set by the anonymous user. To make the sharing simple, the key is given to the user, but is stored separately in a key vault.
- Consequences: Setting up this basic functionality will allow for implementation of authenticated senders and receivers of secrets.
- Stakeholders: Bouvet
- Other possible solutions: Could have provided the user with the key and the one-click link, but since the main focus is implementing sharing secrets with predefined receivers, and Yopass already has this functionality, it was not included.

A.16 Choice of domain model

- Date: 11.01.2023
- Status: Accepted
- Context: The domain model describes the basic functionality which is needed to implement the secret sharing solution.
- Decision: the domain model, seen in Figure 4.1.
- Consequences: The domain model is important, because it describes the main functionality of the application. A poor domain model could lead to poor performance or security issues.
- Stakeholders: Bouvet
- Other possible solutions: Any other version of a domain model which solves the problem at hand.

A.17 Choice of maximum limit of lifetime of blobs

- Date: 11.01.2023
- Status: Accepted
- Context: The maximum lifetime of a blob needs to be set.
- Decision: The maximum allowed lifetime of a blob is 24 hours. A rule has been set for the blob containers which stores secrets, which deletes any blob 24 hours after creation.
- Consequences: Higher security by limiting blob lifetime.
- Stakeholders: Bouvet
- Other possible solutions: No maximum limit of lifetime, or any other value deemed fit.

A.18 Choice of periodic scan and deletion of expired secrets handled by an Azure Function timer trigger

- Date: 11.01.2023
- Status: Accepted
- Context: When a secret's lifetime is reached, and the secret has not yet been accessed, the secret and it's key should be deleted from blob storage and key vault, respectively.
- Decision: Create a separate function application timer trigger which controls the expiration time and date of all secrets currently stored, and deletes secrets which have expired.
- Consequences: Making sure the secret lifetime is upheld.
- Stakeholders: Bouvet
- Other possible solutions: Only utilizing maximum lifetime rule of blobs.

A.19 Choice of ID and URL generation using C# GUID class

- Date: 27.01.2023
- Status: Accepted
- Context: Need a way to connect the secret URL with key and stored secret.
- Decision: The URL used to access a secret will be the same GUID ID used to identify secrets in blob storage and keys in key vault.
- Consequences: Easy and random ID and URL generation and connection to the stored key and secret.
- Stakeholders: Bouvet
- Other possible solutions: Any other type or format for IDs.

A.20 Choice of how to host the Blazor application: Azure web application

- Date: 27.01.2023
- Status: Accepted
- Context: Need to host the application as a resource in Azure.
- Decision: Host it as a web application.
- Consequences: The web application can interact with the storage account and key vault resources in Azure.
- Stakeholders: Bouvet
- Other possible solutions: Any other hosting service.

A.21 Choice of deploying application directly from Visual Studio to Azure

- Date: 27.01.2023
- Status: Accepted
- Context: The application must be deployed to the Azure web application.
- Decision: Deploy directly from Visual Studio.
- Consequences: Not as dynamic as a pipeline/workflow in GitHub, and will be replaced if time allows it.
- Stakeholders: Bouvet
- Other possible solutions: Setting up a GitHub workflow or a CI/CD which publishes the web application following a pull request or a commit to main.

A.22 Choice of authorization protocol: OAuth 2.0 framework

- Date: 06.01.2023
- Status: Accepted
- Context: The web application needs an authorization flow.
- Decision: The OAuth 2.0 framework is chosen, as it is the industry standard [29].
- Consequences: OAuth 2.0 can be combined with OIDC, which is also supported by Azure AD.
- Stakeholders: Bouvet
- Other possible solutions: Any other authorization protocol supported by Azure AD.

A.23 Choice of authentication protocol: OpenId Connect (OIDC) Protocol

- Date: 06.01.2023
- Status: Accepted
- Context: OAuth 2.0 is chosen as the authorization protocol, and another protocol is also needed for authentication.
- Decision: OIDC, a layer on top of OAuth 2.0, is chosen.
- Consequences: Both OAuth 2.0 and OIDC are supported by Azure AD.
- Stakeholders: Bouvet
- Other possible solutions: Any other authentication protocol supported by Azure AD.

A.24 Choice of additional functionality for the sharing solution

- Date: 05.03.2023
- Status: Accepted
- Context: Functionality to be implemented once OAuth 2.0 and OpenId Connect flow has been implemented.
- Decision: A user logs in to the web application, and can choose one or more receiver for their secret. Each receiver can authenticate themselves and view the secret once. Search function for receivers will also be implemented.
- Consequences: Higher security in sharing secrets, the creator of the secret can be sure that a secret can only be accessed the predefined receiver(s).
- Stakeholders: Bouvet
- Other possible solutions: Any other functionality deemed fit.

A.25 Choice of logging of activities

- Date: 05.03.2023
- Status: Accepted
- Context: Need to log activities in the web application and the function application.
- Decision: See the overview in Section 4.9.6.
- Consequences: Non-repudiation of actions.
- Stakeholders: Bouvet
- Other possible solutions: Any other set of logging rules deemed fit.

A.26 Choice of Azure Active Directory (AD) tenant for testing purposes

- Date: 05.03.2023
- Status: Accepted
- Context: For testing purposes a tenant is needed, because administrative privileges needs to be given to the application registration of the Bastion web application.
- Decision: Create an AD tenant for testing purposes, separate from the Bouvet AD tenant.
- Consequences: Secure 'sandbox' for testing and application registration.
- Stakeholders: Bouvet
- Other possible solutions: Could have utilized the Bouvet tenant with administrative approval.

A.27 Choice of application registration roles

- Date: 05.03.2023
- Status: Accepted
- Context: The application registration needs access to data related to the Azure AD users in the tenant created for testing.
- Decision: See roles in Section 4.8.2.
- Consequences: These roles give the web application access to basic user information of all Azure AD users in the tenant.
- Stakeholders: Bouvet
- Other possible solutions: Any other roles deemed fit.

A.28 Choice of OID to identify authenticated users

- Date: 05.03.2023
- Status: Accepted
- Context: Need something to identify the users by when setting sender and receiver for a secret. This information is stored alongside the encrypted message in blob storage.
- Decision: Utilize the unique OID for users in Azure AD.
- Consequences: An unique identifier that cannot be changed.
- Stakeholders: Bouvet
- Other possible solutions: Name or e-mail, but these values can be updated or changed.

A.29 Choice of token validation

- Date: 20.03.2023
- Status: Accepted
- Context: Need to validate tokens to increase security of web application.
- Decision: Validate audience and issuer of token.
- Consequences: Validating tokens mitigates spoofing and other malicious behaviour.
- Stakeholders: Bouvet
- Other possible solutions: No token validation.

A.30 Choice of metadata for storing OID of receivers

- Date: 28.03.2023
- Status: Accepted
- Context: Need to store receivers of secrets somewhere. This was originally planned to be in the text file stored as a blob.
- Decision: Utilize blob metadata key-value pairs to store receivers and their viewer status.
- Consequences: No need to download the blob to be able to read the receivers. Only need the blob name and a blob client, making the operation more effective.
- Stakeholders: Bouvet
- Other possible solutions: The original plan of storing the OIDs in the text file.