blindsend

blindsend is an open source tool for private, end-to-end encrypted file exchange. It supports two use cases - sharing files and requesting files - described below. A demo can be found on the following link: https://blindsend.io

Sharing files

You will start by uploading files and obtaining a link to download. You can then share the link with anyone and they will be able to download the files.
We call this the “Share file” use case.

If you are sharing the link via an unsecure channel like facebook messenger, a password can be set to additionally improve the security. You should share the password using a different secure channel, for example, in person.

Requesting files

If you need someone to send you a file, you will generate a request file link and send it to that person. They will open the link and upload the requested files, after which you can access the same link to download the files.

We call this the “Request file” use case.

Similar to the sharing files use-case, you can set a password which needs to be input when downloading files.

This use-case is suitable for various professional services, such as doctors asking for blood results or a lawyer asking for a subpoena. Traditionally, those documents were shared using an insecure channel such as email.

Architecture

Blindsend consists of four parts:

1. Server, which provides a REST API for managing file exchange workflows.
2. Web UI, which handles encryption and decryption of files on agents' local machines, and provides a web client.
3. Cloud Storage on the Google cloud platform where the encrypted files are stored.
4. PostgreSQL where link data is stored.

Security

Files uploaded to blindsend are encrypted using an end-to-end encrypted protocol, meaning neither blindsend nor any third party can decrypt them.

Only the persons possessing the link (and an optional password) can decrypt the files.
It is important to share the link using an authenticated channel, meaning the link wasn't changed during the transfer and the other party receives the same link you sent them.
If the channel is not secure, a password can be set to prevent the third party from decrypting the files.

To keep sensitive information away from the blindsend servers, links use the URL fragments. They are the parts of the URL after the **#** symbol which are not sent to the server when the URL is opened in the browser.

Sharing files

Protocol for sharing files works as follows.

Uploading files:

1. Sender sets the files and a password. If no password is set, an empty value is used.
2. Random 128-bit seed (link_seed) is generated.
3. Random 128-bit salt is generated.
4. Password hash is generated using PBKDF2 with the password (step 1) and salt (step 3) as inputs.
5. Seed is generated by hashing the concatenated random seed (step 2) and password hash (step 4) using SHA-256.
6. Multiple 256-bit keys are derived using HKDF and seed from step 5:
    a. One to encrypt the metadata, with string ‘metadata’ as context.
    b. One for each file, with file number (starting at 0) as context.
7. Random 96-bit identifications vectors are generated:
    a. One for metadata.
    b. One for each file.
8. Metadata is created for each file, containing file name, size, id and identification vector from step 7 b).
9. All metadata is joined and encrypted using AES-GCM with the key from step 6 a) and IV from step 7 a). IV is concatenated to the encrypted metadata.
10. Following data is submitted to blindsend server:
    a. Encrypted metadata from step 9.
    b. Hashed seed from step 5 (additionally hashed with SHA-256).
    c. Salt from step 3.
    d. File ids.

    Server responds with link id.
11. Each file is split into chunks of 4MB and for each chunk:
    a. IV is generated as a SHA-256 hash of concatenated file IV from step 7 b) and chunk id (starting from 0).
    b. Chunk is encrypted using AES-GCM with the key from step 6 b) and IV from step 11 a). IV is concatenated to the encrypted chunk.
    c. Encrypted chunks are concurrently uploaded and assembled on the server.
12. After all files are uploaded, a link is generated containing link id from step 10 and seed from step 2.
    https://blindsend.io#{link_id};{seed_link}

Downloading files:

1. Link id and seed are extracted from the URL fragment.
2. Using the link id, from server are requested:
    a. Encrypted metadata
    b. Hashed seed
    c. Salt
3. User submits the password.
4. Password hash is generated using PBKDF2 with the password (step 3) and salt (step 2 c) as inputs.
5. Seed is generated by hashing the concatenated seed from the URL fragment (step 1) and password hash (step 4) using SHA-256.
6. The seed from step 5 is hashed with SHA-256 again and compared to the value received from the server (step 2 b).
    a. If they differ, the password is wrong given no other data was manipulated.
7. Metadata key is derived using HKDF with seed from step 5 and string ‘metadata’ as context.
8. Encrypted metadata is decrypted using AES-GCM with the key from step 7.
9. For each file specified in the metadata from step 8, a key is derived using the HKDF with seed from step 5 and with file number (starting at 0) as context.
10. A File is decrypted. While downloading, the file stream is transformed into chunks of 4MB and for each chunk:
    a. IV is generated as a SHA-256 hash of concatenated file IV (first 12 bytes of the encrypted data) and chunk id (starting from 0).
    b. Chunk is decrypted using AES-GCM with the key from step 9 and IV from step 10 a).
    c. Decrypted stream is concatenated to the previous chunk one until the whole file is decrypted.
11. If there are multiple files, decrypted streams are piped to a zip32 transformer so they can be downloaded as an archive.

Requesting files

2 parties are distinguished:

- File requester who generates a link and downloads files
- File uploader who uses the link to upload files

Protocol for requesting files works as follows.

Generating upload link:

1. User sets a password. If no password is set, an empty value is used.
2. Random 128-bit salt is generated.
3. A 256-bit key is derived using PBKDF2 with the password from step 1 and salt from step 2.
4. Random elliptic curve (P-256) key pair is generated.
5. Secret key from step 4 is encrypted using AES-GCM with the key from step 3.
6. Public key from step 4 is hashed using SHA-256.
7. Following data is submitted to blindsend server:
    a. Public key from step 4.
    b. Salt from step 2.
    c. Encrypted secret key from step 4.

    Server responds with link id.
8. A link is generated containing link id from step 7 and hashed public key from step 6.
   https://blindsend.io#{link_id};{hashed_public_key}

Uploading files:

1. Link id and hashed public key are extracted from the URL fragment.
2. Using the link id, the requester's public key is fetched from the server.
3. Requesters' public key is hashed using SHA-256 and compared to the hashed public key from step 1.
    a. If they differ, data was manipulated and the process stops.
4. Random elliptic curve (P-256) key pair is generated.
5. Seed is derived using ECDH with the requester’s public key form step 2 and uploader’s key pair from step 4.
6. Multiple 256-bit keys are derived using HKDF and seed from step 5:
    a. One to encrypt the metadata, with string ‘metadata’ as context.
    b. One for each file, with file number (starting at 0) as context.
7. Random 96-bit identifications vectors are generated:
    a. One for metadata.
    b. One for each file.
8. Metadata is created for each file, containing file name, size, id and identification vector from step 7 b).
9. All metadata is joined and encrypted using AES-GCM with the key from step 6 a) and IV from step 7 a). IV is concatenated to the encrypted metadata.
10. Following data is submitted to blindsend server:
    a. Encrypted metadata from step 9.
    b. Hashed seed from step 5 (additionally hashed with SHA-256).
    c. Public key from step 4.
11. Each file is split into chunks of 4MB and for each chunk:
    a. IV is generated as a SHA-256 hash of concatenated file IV from step 7 b) and chunk id (starting from 0).
    b. Chunk is encrypted using AES-GCM with the key from step 6 b) and IV from step 11 a). IV is concatenated to the encrypted chunk.
    c. Encrypted chunks are concurrently uploaded and assembled on the server.

Downloading files:

1. Link id is extracted from the URL fragment.
2. Using the link id, from server are requested:
    a. Encrypted metadata
    b. Hashed seed
    c. Salt
    d. Uploader’s public key
    e. Encrypted requester’s secret key
3. User submits the password.
4. A 256-bit key is derived using PBKDF2 with the password from step 3 and salt from step 2.
5. Encrypted secret key obtained in step 2 is decrypted using the key from step 4.
6. Seed is derived using ECDH with the uploader’s public key form step 2 and requester’s secret key from step 5.
7. Seed from step 6 is hashed using SHA-256 and compared with hashed seed obtained in step 2.
    a. If they differ, the password is wrong given no other data was manipulated.
8. Metadata key is derived using HKDF with seed from step 6 and string ‘metadata’ as context.
9. Encrypted metadata is decrypted using AES-GCM with the key from step 7.
10. For each file specified in the metadata from step 9, a key is derived using the HKDF with seed from step 6 and with file number (starting at 0) as context.
11. A File is decrypted. While downloading, the file stream is transformed into chunks of 4MB and for each chunk:
    a. IV is generated as a SHA-256 hash of concatenated file IV (first 12 bytes of the encrypted data) and chunk id (starting from 0).
    b. Chunk is decrypted using AES-GCM with the key from step 9 and IV from step 10 a).
    c. Decrypted stream is concatenated to the previous chunk one until the whole file is decrypted.
12. If there are multiple files, decrypted streams are piped to a zip32 transformer so they can be downloaded as an archive.