Blinded Unique Identifiers

A **privacy preserving** mechanism to find associated domains **across actors**

Example

```
$ whois kumari.net

Domain Name: kumari.net
Updated Date: 2026-01-12T15:11:56Z
Creation Date: 2014-11-12T11:53:15Z
Registry Expiry Date: 2035-11-12T11:53:15Z

Registrant Name: Warren Kumari
Registrant Organization: Acme Anvils
Registrant Phone: +1-555-867-5309
Registrant Email: warren@kumari.net
```

Example

```
$ whois kumari.net

Domain Name: kumari.net
Updated Date: 2026-01-12T15:11:56Z
Creation Date: 2014-11-12T11:53:15Z
Registry Expiry Date: 2035-11-12T11:53:15Z

[redacted]
```

Example

```
$ whois kumari.net

Domain Name: kumari.net
Updated Date: 2026-01-12T15:11:56Z
Creation Date: 2014-11-12T11:53:15Z
Registry Expiry Date: 2035-11-12T11:53:15Z

Registrant Name: BUID-cb42452d14e030bbe8a7604e1990cbcb394f0481
Registrant Organization: BUID-f0ec0188fc229e77e65418813f1e1dc7f3d9ee54
Registrant Phone: BUID-a53085e823f9c471e613d78255cdeded9d1cf53c
Registrant Email: BUID-245b72ebbd0dad12fa5fd16eecff8c6e6aa0ee5d
```

Example

```
$ whois kumari.net

Domain Name: kumari.net
Updated Date: 2026-01-12T15:11:56Z
Creation Date: 2014-11-12T11:53:15Z
Registry Expiry Date: 2035-11-12T11:53:15Z

Registrant Name: BUID-cb42452d14e030bbe8a7604e1990cbcb394f0481
Registrant Organization: BUID-f0ec0188fc229e77e65418813f1e1dc7f3d9ee54
Registrant Phone: BUID-a53085e823f9c471e613d78255cdeded9d1cf53c
Registrant Email: BUID-245b72ebbd0dad12fa5fd16eecff8c6e6aa0ee5d
```

Cyber attack icons created by nangicon - Flaticon

evil.com
bad.net
bank-phish.io
hakr@aol.com

Registrar A

Cyber attack icons created by nangicon - Flaticon

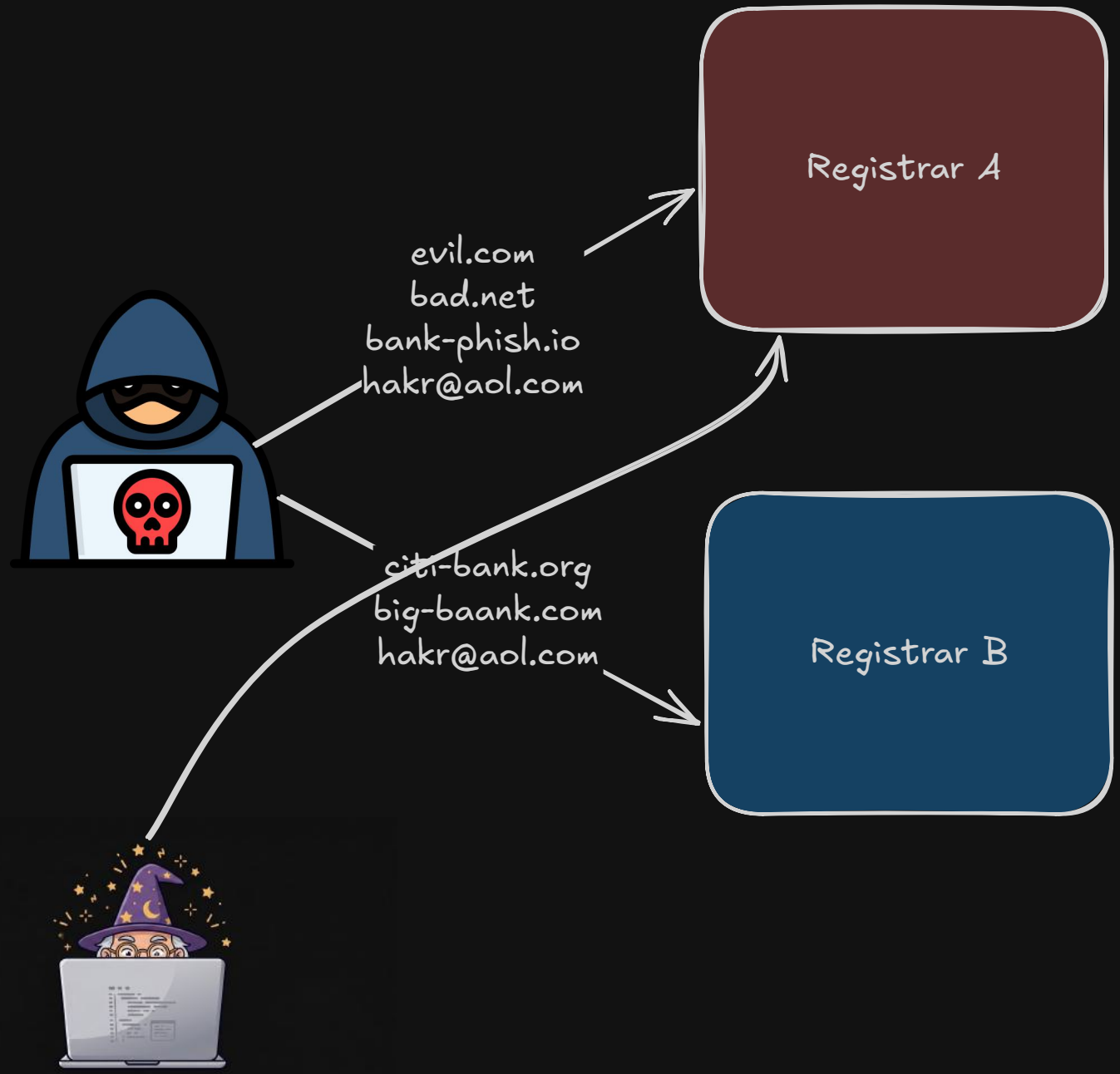
evil.com
bad.net
bank-phish.io
hakr@aol.com

citi-bank.org
big-baank.com
hakr@aol.com

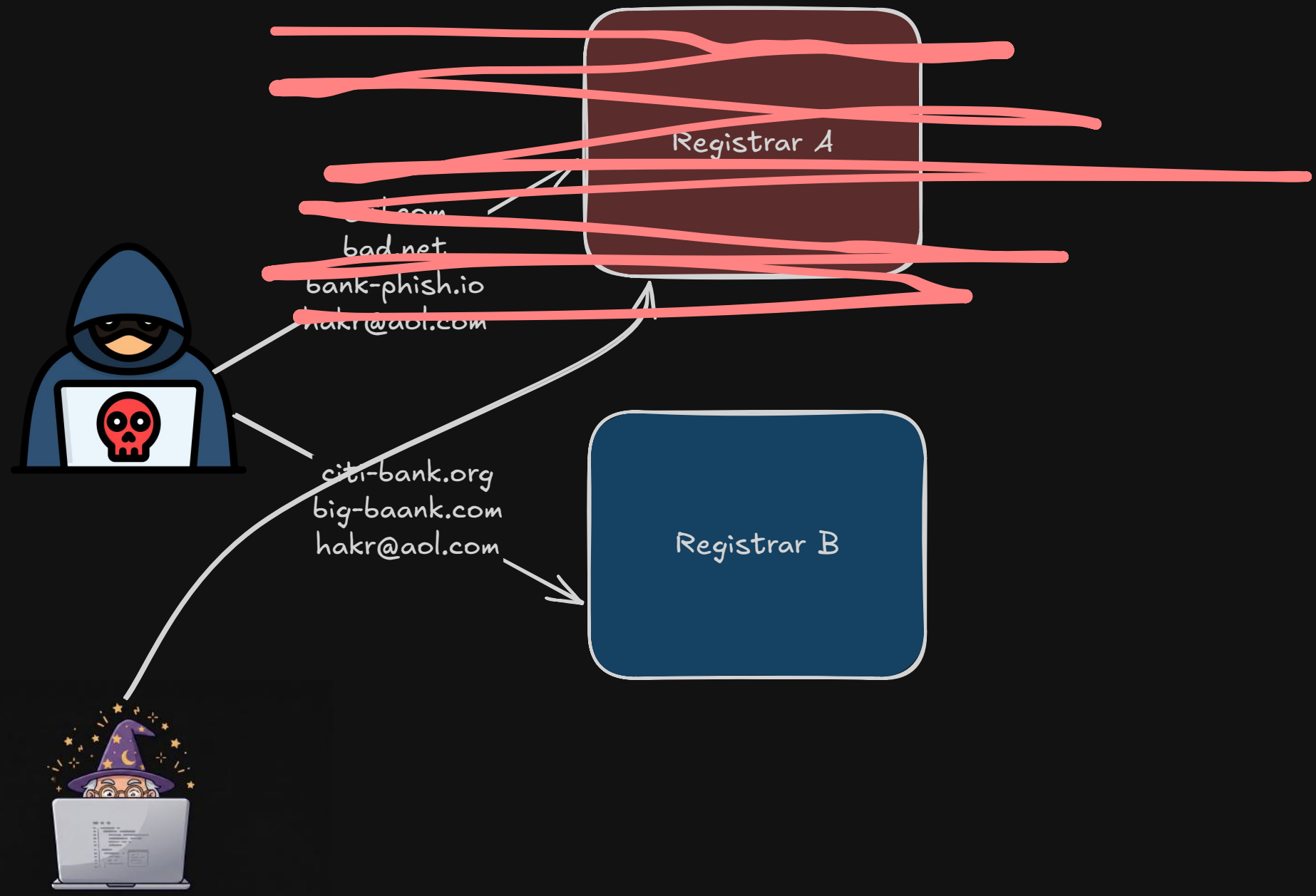
Registrar A

Registrar B

Cyber attack icons created by nangicon - Flaticon

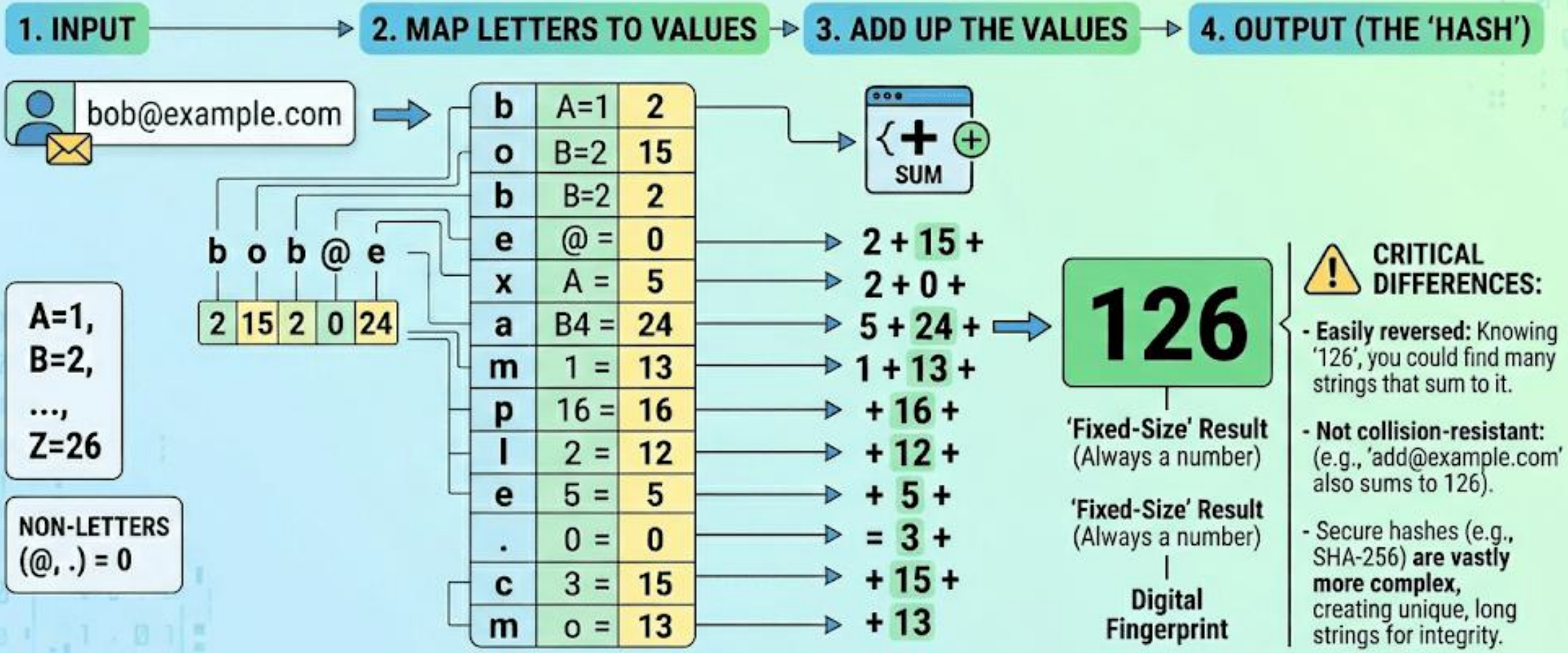


Cyber attack icons created by nangicon - Flaticon

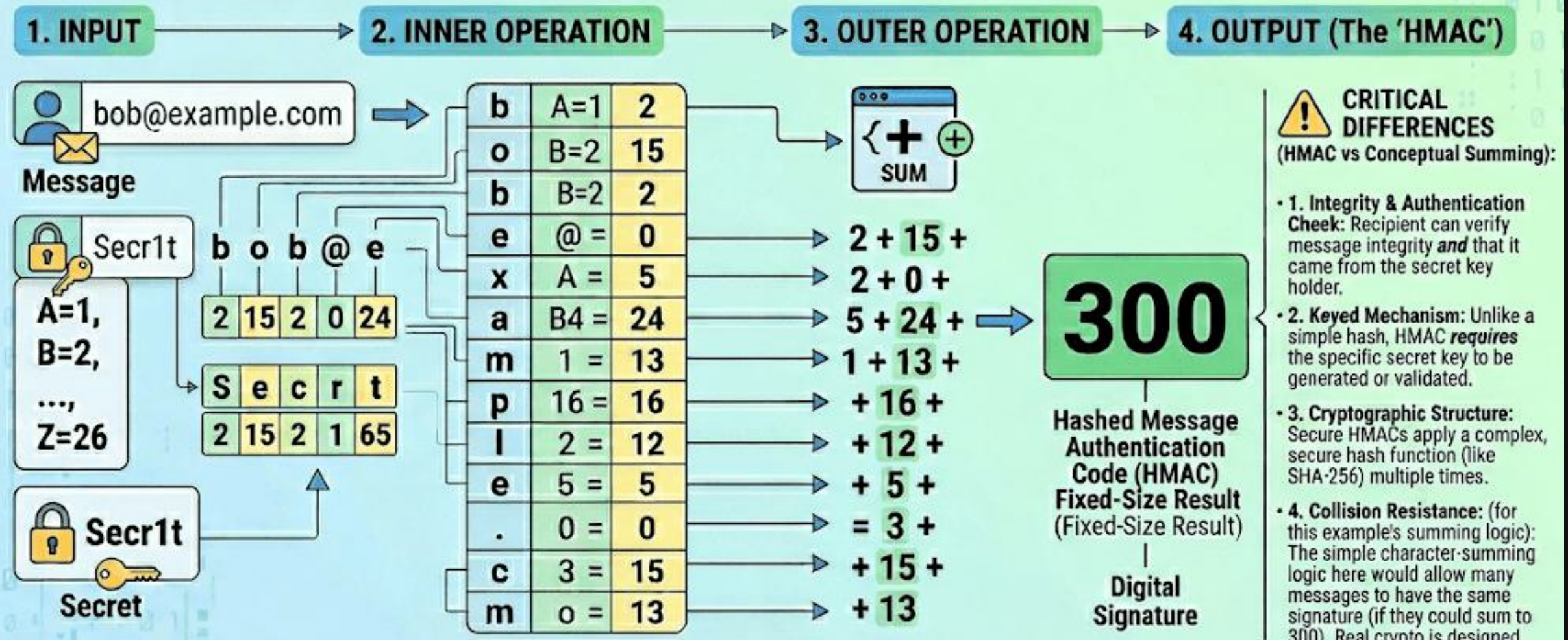


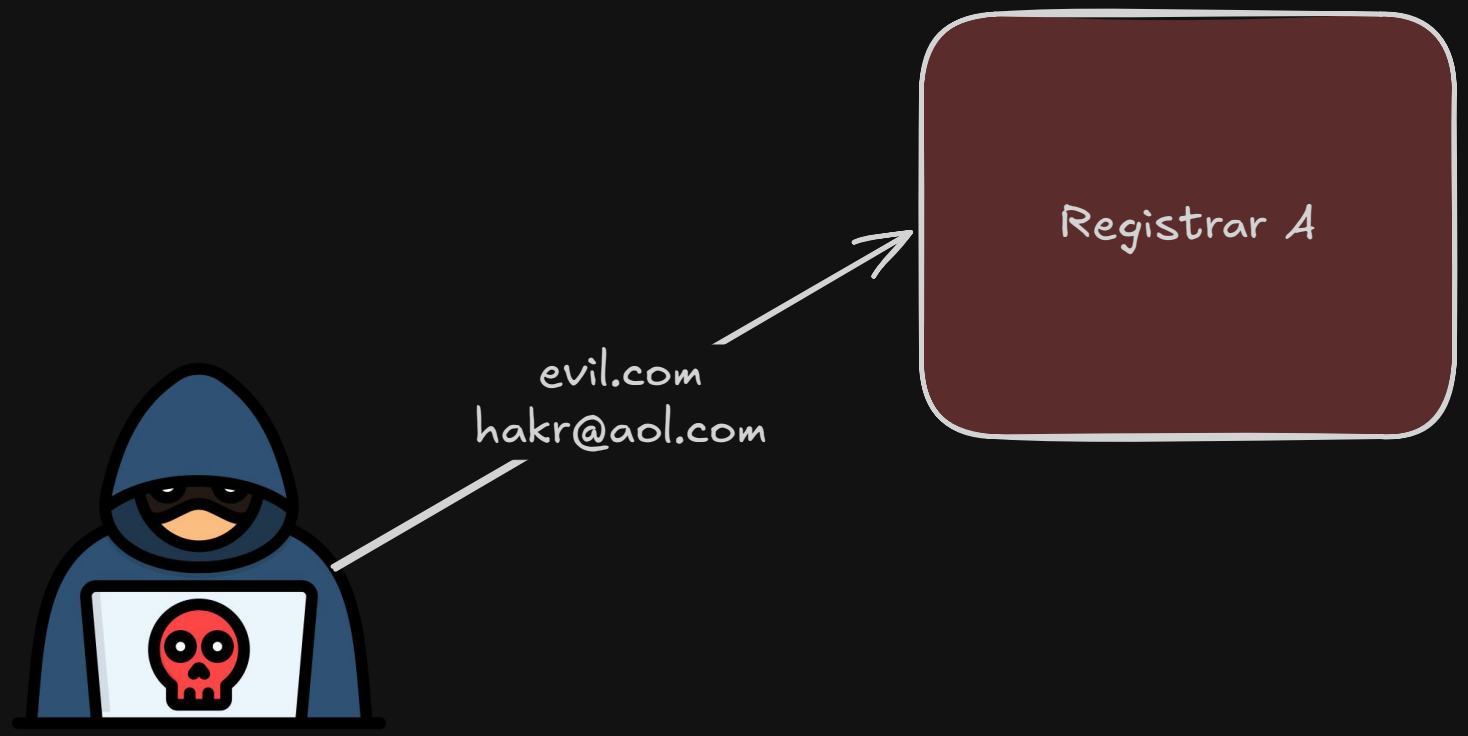
Cyber attack icons created by nangicon - Flaticon

Making a hash of things...

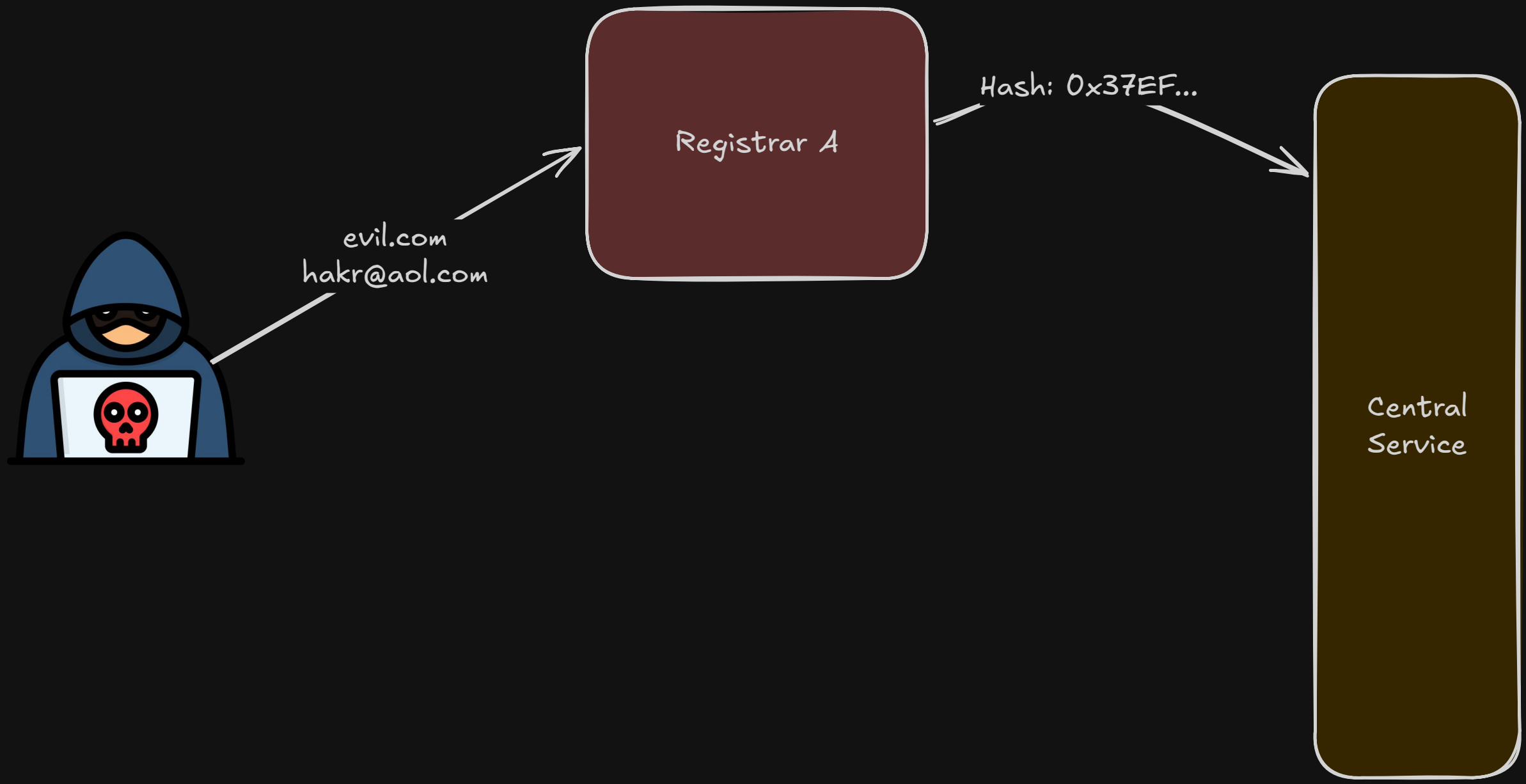


HMACs - basically just a hash and a secret...

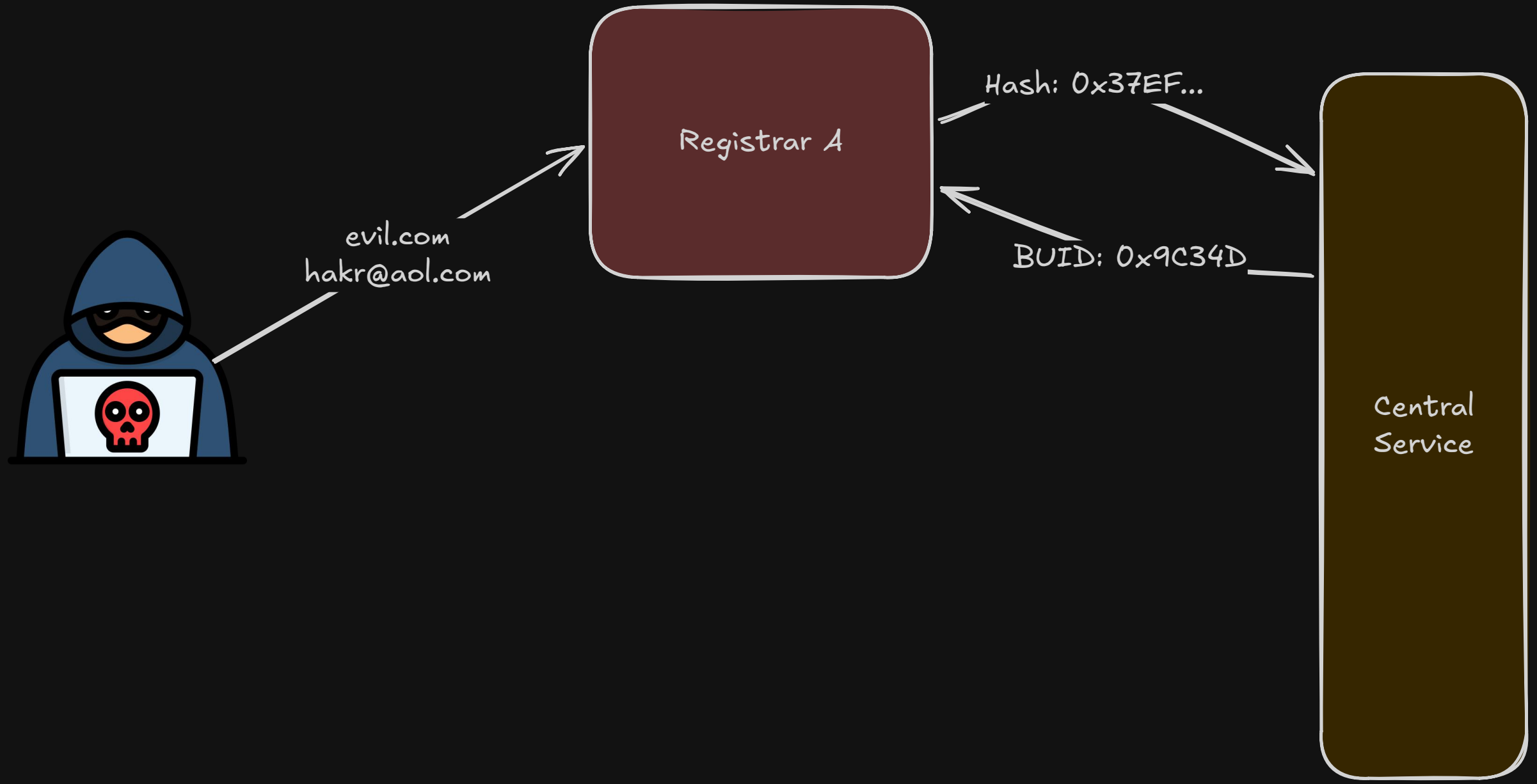




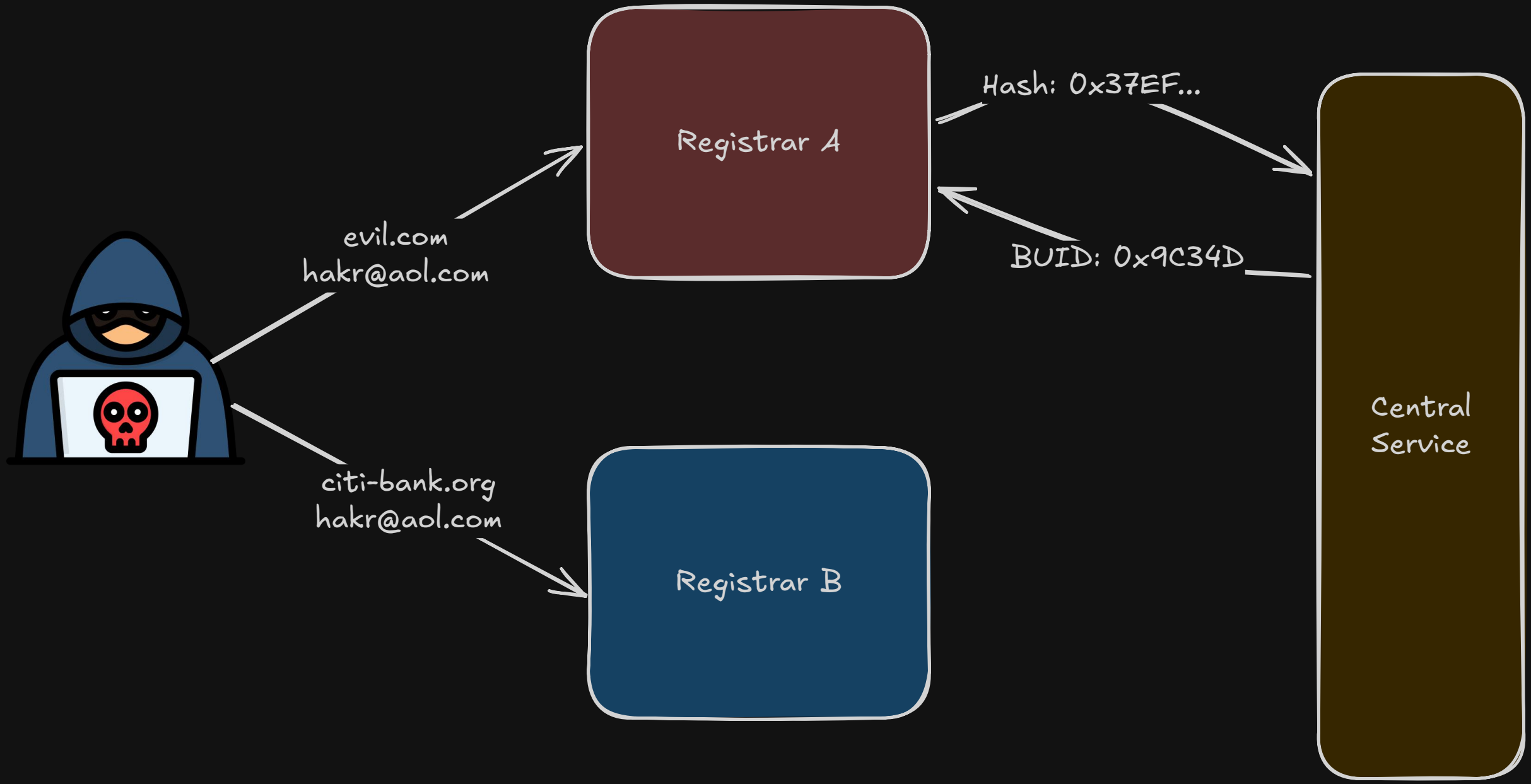
Cyber attack icons created by nangicon - Flaticon



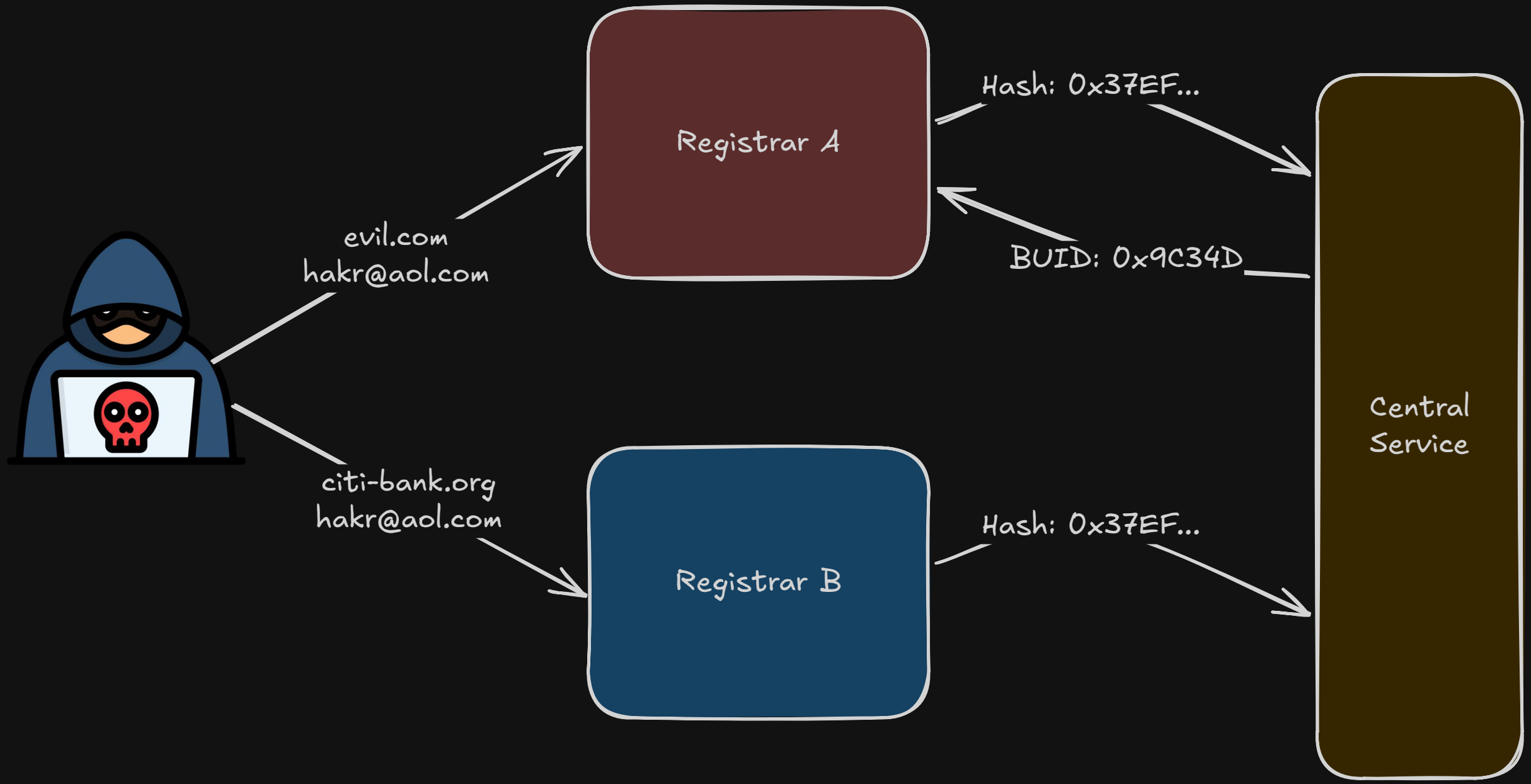
Cyber attack icons created by nangicon - Flaticon



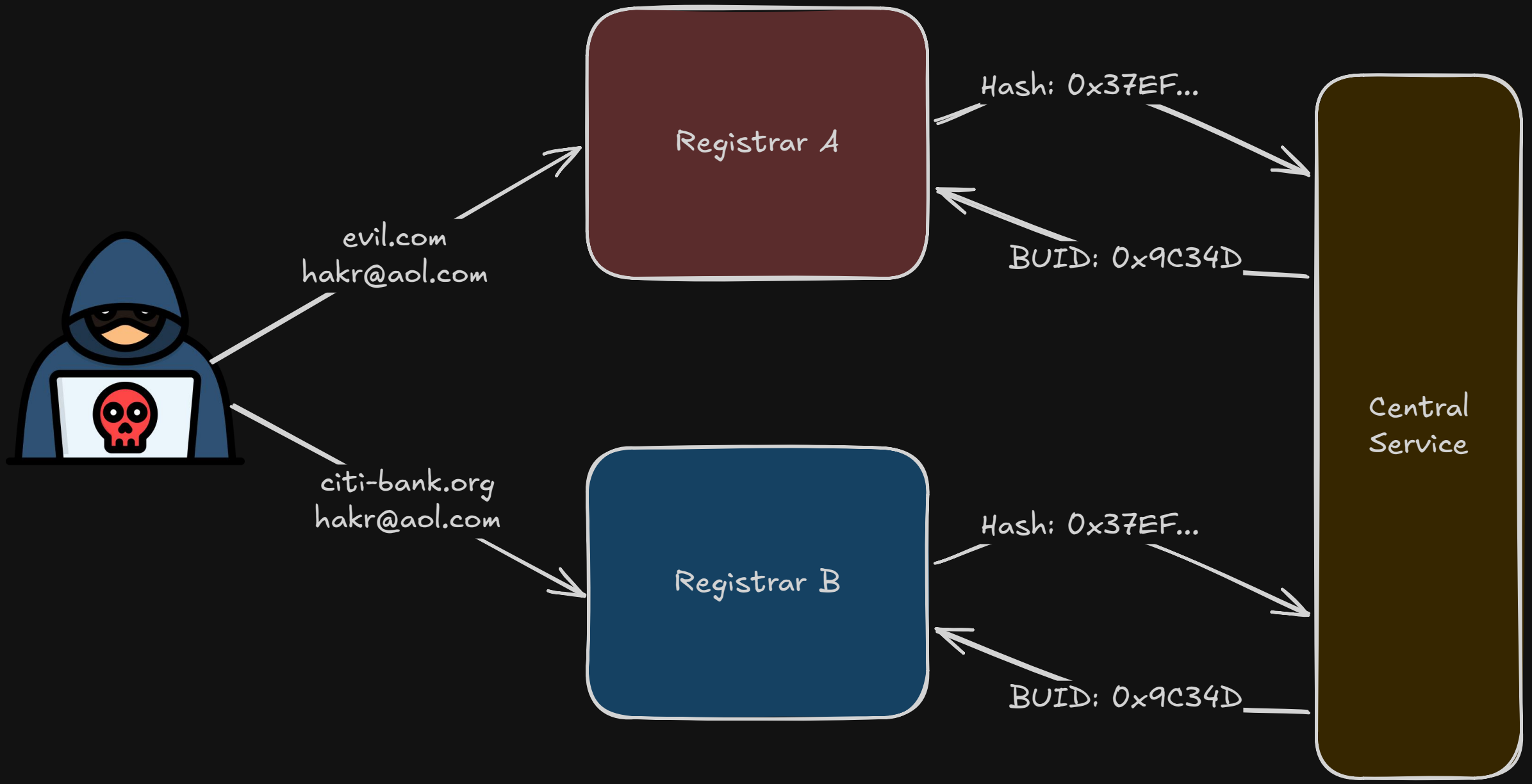
Cyber attack icons created by nangicon - Flaticon



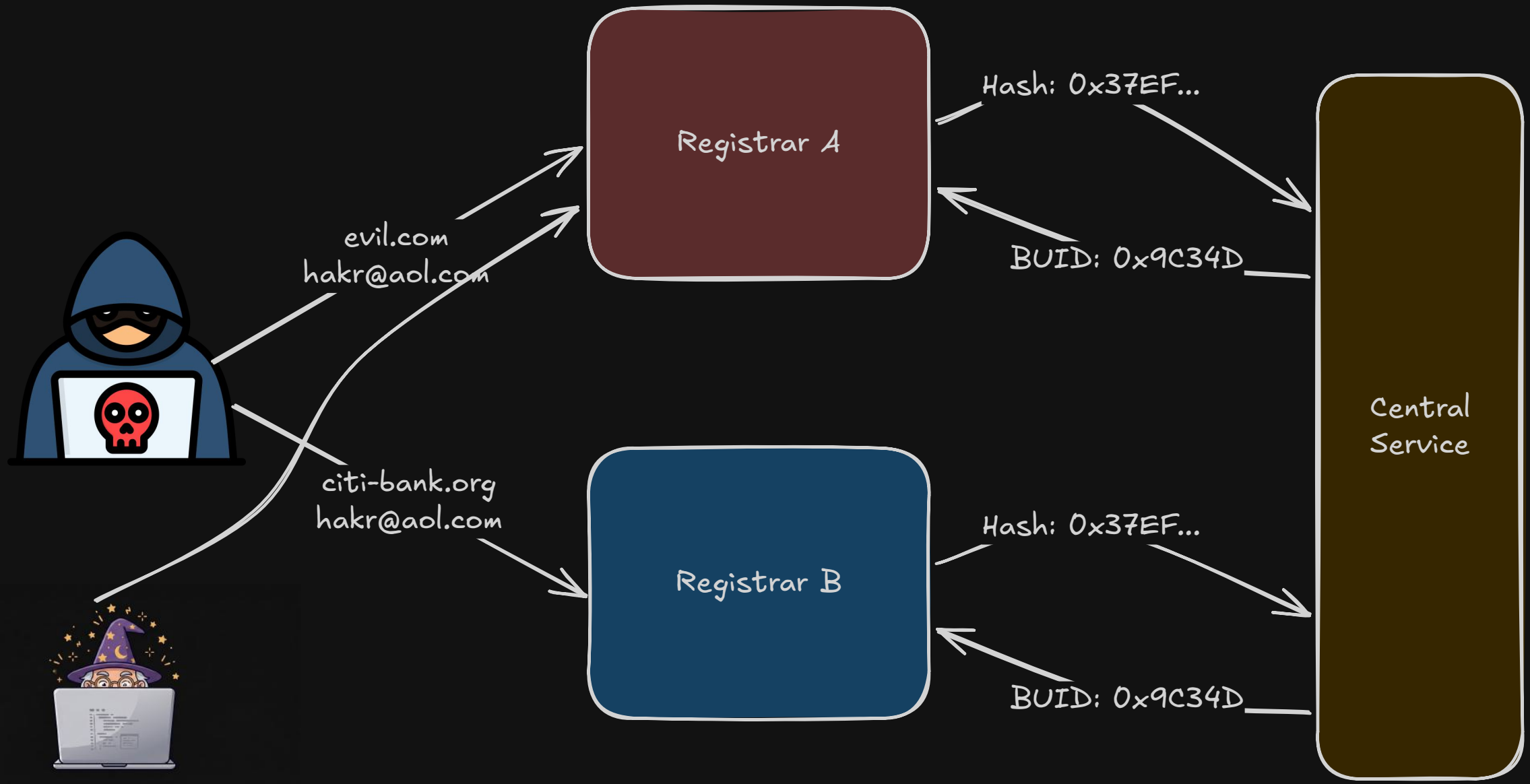
Cyber attack icons created by nangicon - Flaticon



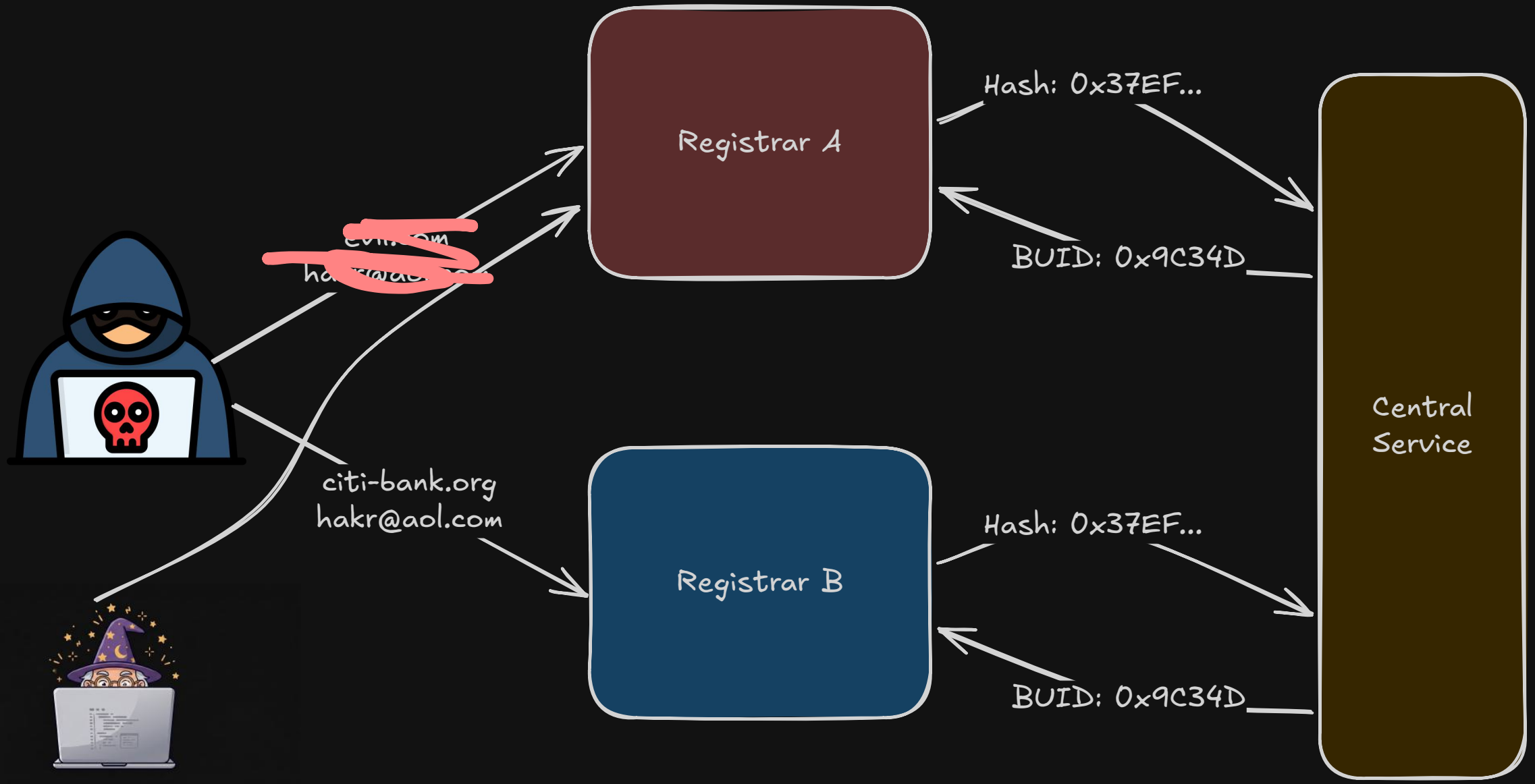
Cyber attack icons created by nangicon - Flaticon



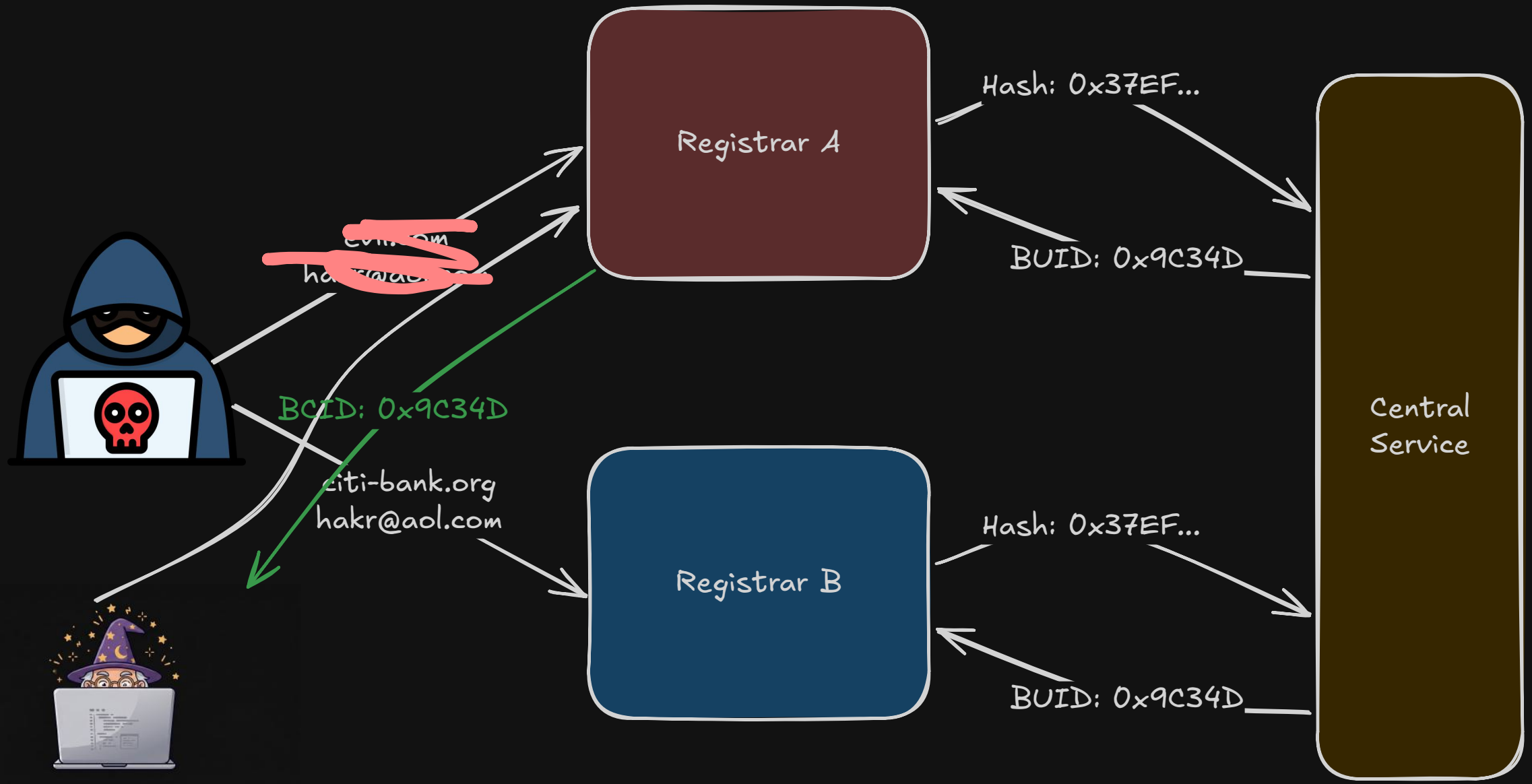
Cyber attack icons created by nangicon - Flaticon



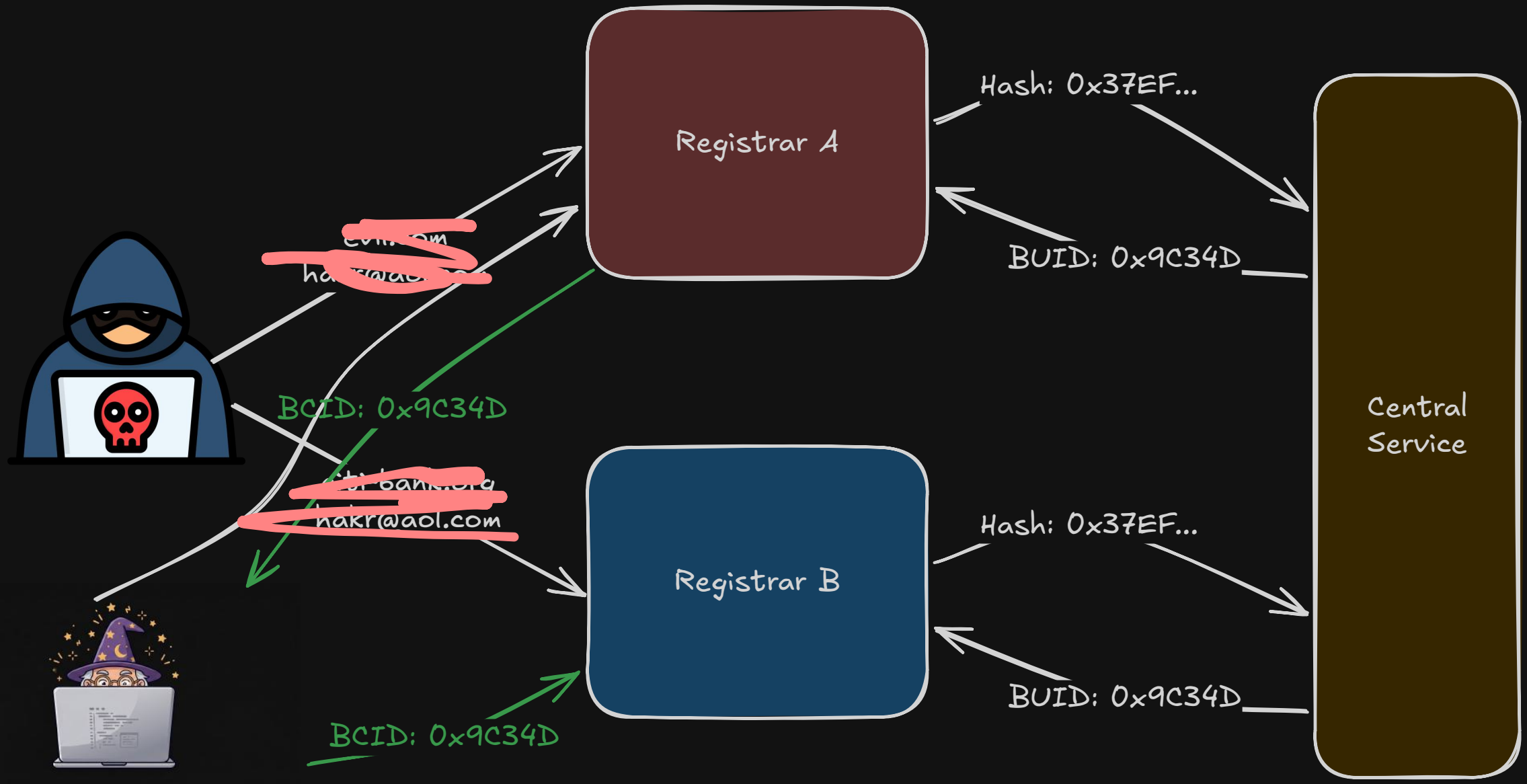
Cyber attack icons created by nangicon - Flaticon



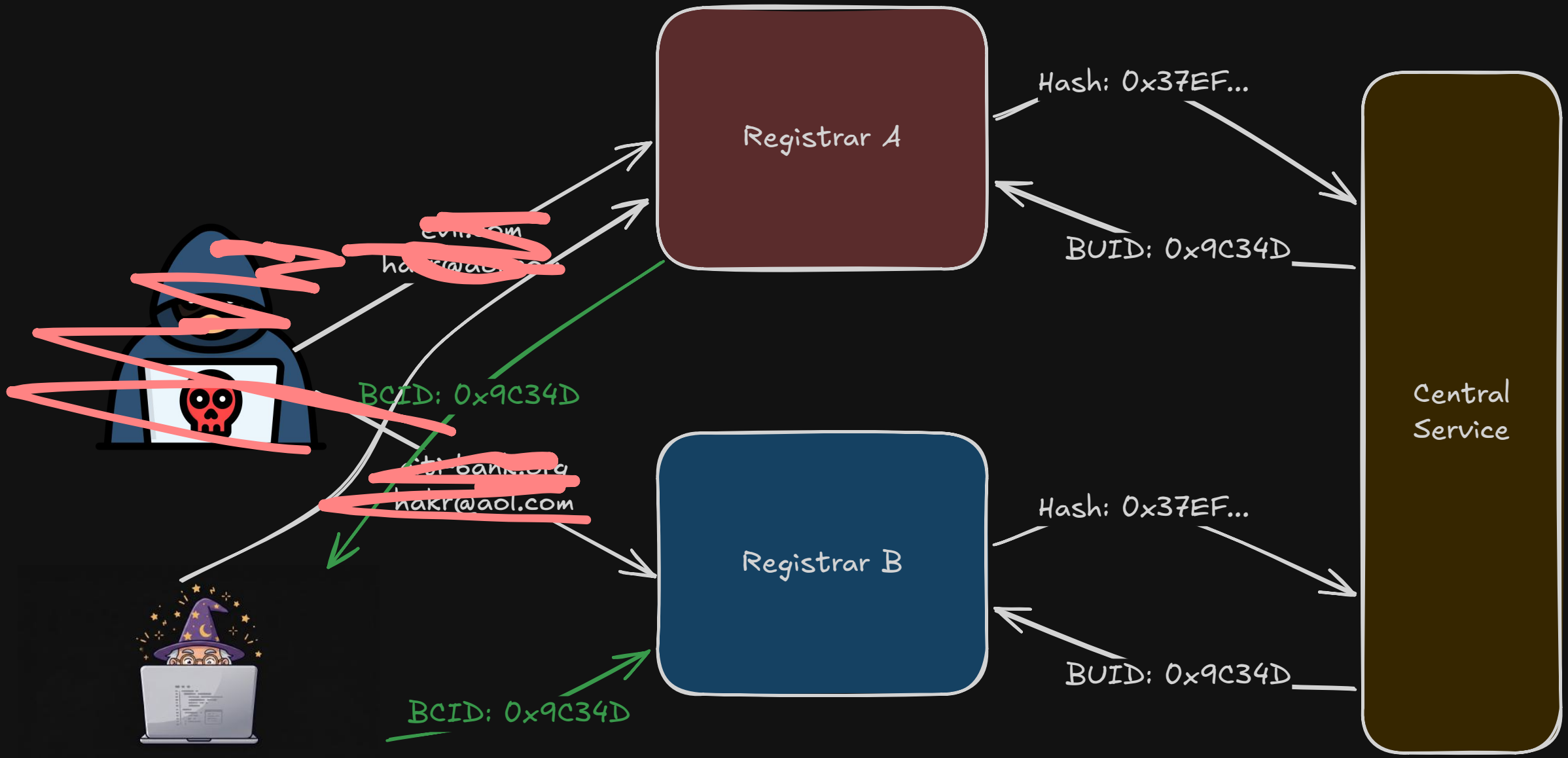
Cyber attack icons created by nangicon - Flaticon



Cyber attack icons created by nangicon - Flaticon



Cyber attack icons created by nangicon - Flaticon



Cyber attack icons created by nangicon - Flaticon

Considerations / FAQ

Considerations / FAQ

- Why not just publish the hash instead of the BUID?!

Considerations / FAQ

- Why not just publish the hash instead of the BUID?!
  - Dictionary attack: a@a.com, b@a.com, zzzzz@zzzz.com

Considerations / FAQ

- Why not just publish the hash instead of the BUID?!
  - Dictionary attack: a@a.com, b@a.com, zzzzz@zzzz.com
- Why not just send the raw data to the central service?!

Considerations / FAQ

- Why not just publish the hash instead of the BUID?!
  - Dictionary attack: a@a.com, b@a.com, zzzzz@zzzz.com
- Why not just send the raw data to the central service?!
  - Don't necessarily want to expose my customer info to the service.

Considerations / FAQ

- Why not just publish the hash instead of the BUID?!
  - Dictionary attack: a@a.com, b@a.com, zzzzz@zzzz.com
- Why not just send the raw data to the central service?!
  - Don't necessarily want to expose my customer info to the service.
- ... but the service could also do a dictionary attack!

Considerations / FAQ

- Why not just publish the hash instead of the BUID?!
  - Dictionary attack: a@a.com, b@a.com, zzzzz@zzzz.com
- Why not just send the raw data to the central service?!
  - Don't necessarily want to expose my customer info to the service.
- ... but the service could also do a dictionary attack!
  - Yes, they could. But we are not treating the central service as fully malicious.

Considerations / FAQ

- Why not just publish the hash instead of the BUID?!
  - Dictionary attack: a@a.com, b@a.com, zzzzz@zzzz.com
- Why not just send the raw data to the central service?!
  - Don't necessarily want to expose my customer info to the service.
- ... but the service could also do a dictionary attack!
  - Yes, they could. But we are not treating the central service as fully malicious.
- Bob could just change his e.g email - bob@example.com, BoB@example.com

Considerations / FAQ

- Why not just publish the hash instead of the BUID?!
  - Dictionary attack: a@a.com, b@a.com, zzzzz@zzzz.com
- Why not just send the raw data to the central service?!
  - Don't necessarily want to expose my customer info to the service.
- ... but the service could also do a dictionary attack!
  - Yes, they could. But we are not treating the central service as fully malicious.
- Bob could just change his e.g email - bob@example.com, BoB@example.com
  - Yup. Firstly canonicalize the raw data.

Considerations / FAQ

- Why not just publish the hash instead of the BUID?!
  - Dictionary attack: a@a.com, b@a.com, zzzzz@zzzz.com
- Why not just send the raw data to the central service?!
  - Don't necessarily want to expose my customer info to the service.
- ... but the service could also do a dictionary attack!
  - Yes, they could. But we are not treating the central service as fully malicious.
- Bob could just change his e.g email - bob@example.com, BoB@example.com
  - Yup. Firstly canonicalize the raw data.
- Fine! bob@example.com, bob1@example.com

Considerations / FAQ

- Why not just publish the hash instead of the BUID?!
  - Dictionary attack: a@a.com, b@a.com, zzzzz@zzzz.com
- Why not just send the raw data to the central service?!
  - Don't necessarily want to expose my customer info to the service.
- ... but the service could also do a dictionary attack!
  - Yes, they could. But we are not treating the central service as fully malicious.
- Bob could just change his e.g email - bob@example.com, BoB@example.com
  - Yup. Firstly canonicalize the raw data.
- Fine! bob@example.com, bob1@example.com
  - Yup. These are different, and correctly create different BUIDs.

Considerations / FAQ

- Why not just publish the hash instead of the BUID?!
  - Dictionary attack: a@a.com, b@a.com, zzzzz@zzzz.com
- Why not just send the raw data to the central service?!
  - Don't necessarily want to expose my customer info to the service.
- ... but the service could also do a dictionary attack!
  - Yes, they could. But we are not treating the central service as fully malicious.
- Bob could just change his e.g email - bob@example.com, BoB@example.com
  - Yup. Firstly canonicalize the raw data.
- Fine! bob@example.com, bob1@example.com
  - Yup. These are different, and correctly create different BUIDs.
  - Also create for other info, like Credit Card, Phone, etc etc etc.

Considerations / FAQ

- Why not just publish the hash instead of the BUID?!
  - Dictionary attack: a@a.com, b@a.com, zzzzz@zzzz.com
- Why not just send the raw data to the central service?!
  - Don't necessarily want to expose my customer info to the service.
- ... but the service could also do a dictionary attack!
  - Yes, they could. But we are not treating the central service as fully malicious.
- Bob could just change his e.g email - bob@example.com, BoB@example.com
  - Yup. Firstly canonicalize the raw data.
- Fine! bob@example.com, bob1@example.com
  - Yup. These are different, and correctly create different BUIDs.
  - Also create for other info, like Credit Card, Phone, etc etc etc.
- What if Bob changes everything? Name, Address, CC, Telephone?!

Considerations / FAQ

- Why not just publish the hash instead of the BUID?!
  - Dictionary attack: a@a.com, b@a.com, zzzzz@zzzz.com
- Why not just send the raw data to the central service?!
  - Don't necessarily want to expose my customer info to the service.
- ... but the service could also do a dictionary attack!
  - Yes, they could. But we are not treating the central service as fully malicious.
- Bob could just change his e.g email - bob@example.com, BoB@example.com
  - Yup. Firstly canonicalize the raw data.
- Fine! bob@example.com, bob1@example.com
  - Yup. These are different, and correctly create different BUIDs.
  - Also create for other info, like Credit Card, Phone, etc etc etc.
- What if Bob changes everything? Name, Address, CC, Telephone?!
  - Well, then they are not associated domains...

Self exposure...

Self exposure...

- Bob registers many different attack domains (evil.com, phish.net), as well as a personal domain (hacker-bob.net)

Self exposure...

- Bob registers many different attack domains (evil.com, phish.net), as well as a personal domain (hacker-bob.net)
- On hacker-bob.net he publishes his real name, and photo.

Self exposure...

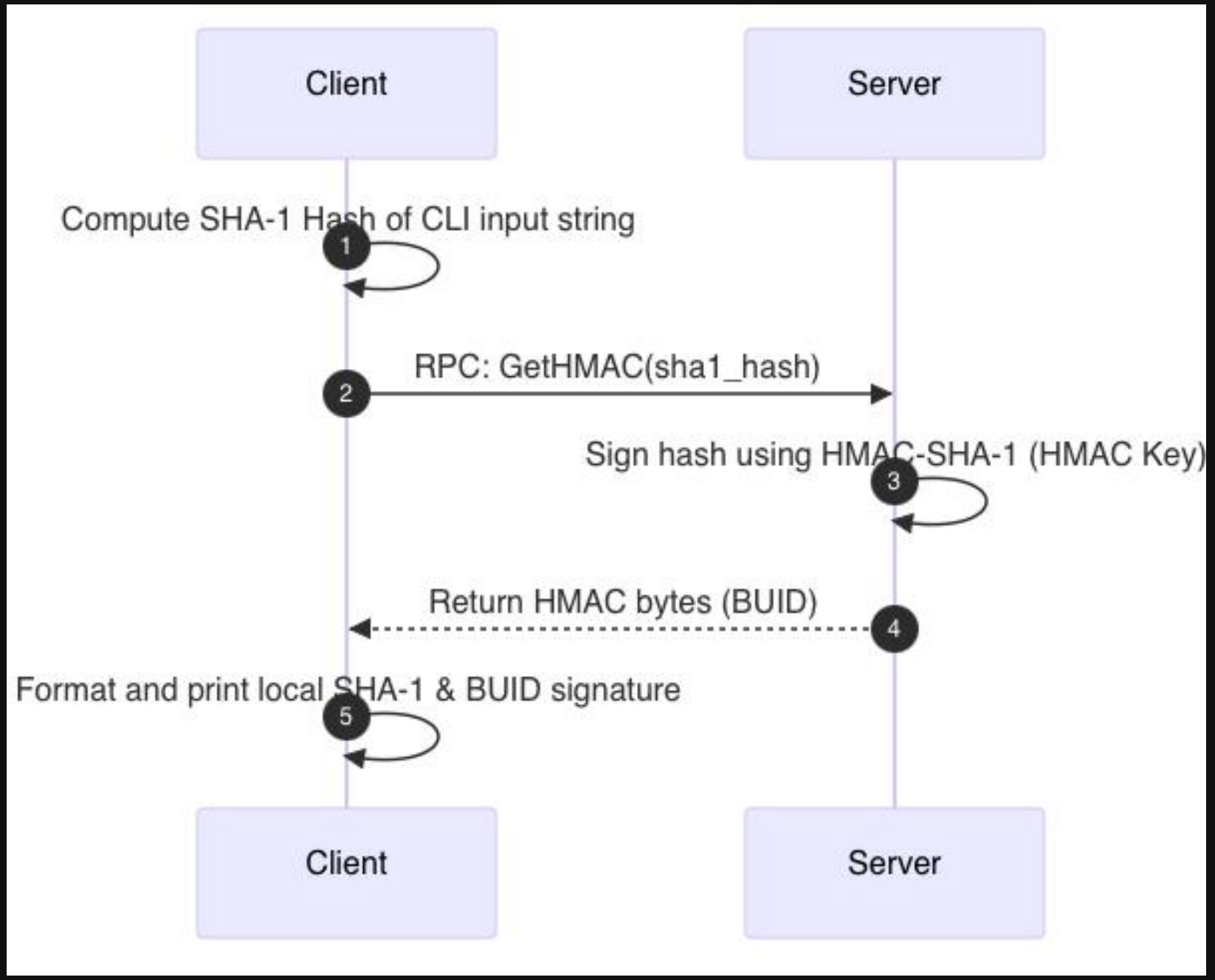
- Bob registers many different attack domains (evil.com, phish.net), as well as a personal domain (hacker-bob.net)
- On hacker-bob.net he publishes his real name, and photo.
- Now a security researcher finds evil.com, gets the associated domains, sees hacker-bob.net. Researches now knows that all of these are related to Bob.

Self exposure...

- Bob registers many different attack domains (evil.com, phish.net), as well as a personal domain (hacker-bob.net)
- On hacker-bob.net he publishes his real name, and photo.
- Now a security researcher finds evil.com, gets the associated domains, sees hacker-bob.net. Researches now knows that all of these are related to Bob.
    - Yes. That's true. That's a function of associated domains...

Costs and complexity...

- Ok, but but someone would need to actually build and this service. That sounds expensive!
  - It's actually not hard at all
  - Demo implementation.


Client
Server
Compute SHA-1 Hash of CLI input string
1
RPC: GetHMAC(sha1_hash)
2
Sign hash using HMAC-SHA-1 (HMAC Key)
3
Return HMAC bytes (BUID)
4
Format and print local SHA-1 & BUID signature
5
Client
Server

Messages

```proto
syntax = "proto3";

package pb;

option go_package = "kumari.net/blinded_unique_identifers/pb";

// IdentifierService defines a service for computing HMAC signatures of client hashes.
service IdentifierService {
  // GetHMAC receives a SHA-1 hash and returns its HMAC-SHA-1 signature.
  rpc GetHMAC(HMACRequest) returns (HMACResponse);
}

// HMACRequest carries the SHA-1 hash of a data segment.
message HMACRequest {
  // The SHA-1 hash of the input data. SHA-1 hash is strictly 20 bytes.
  bytes sha1_hash = 1;
}

// HMACResponse carries the computed HMAC-SHA-1 signature.
message HMACResponse {
  // The HMAC-SHA-1 signature of the SHA-1 hash.
  bytes hmac = 1;
}
```

Messages

```proto
syntax = "proto3";

package pb;

option go_package = "kumari.net/blinded_unique_identifers/pb";

// IdentifierService defines a service for computing HMAC signatures of client hashes.
service IdentifierService {
  // GetHMAC receives a SHA-1 hash and returns its HMAC-SHA-1 signature.
  rpc GetHMAC(HMACRequest) returns (HMACResponse);
}

// HMACRequest carries the SHA-1 hash of a data segment.
message HMACRequest {
  // The SHA-1 hash of the input data. SHA-1 hash is strictly 20 bytes.
  bytes sha1_hash = 1;
}

// HMACResponse carries the computed HMAC-SHA-1 signature.
message HMACResponse {
  // The HMAC-SHA-1 signature of the SHA-1 hash.
  bytes hmac = 1;
}
```

Messages

```proto
syntax = "proto3";

package pb;

option go_package = "kumari.net/blinded_unique_identifers/pb";

// IdentifierService defines a service for computing HMAC signatures of client hashes.
service IdentifierService {
  // GetHMAC receives a SHA-1 hash and returns its HMAC-SHA-1 signature.
  rpc GetHMAC(HMACRequest) returns (HMACResponse);
}

// HMACRequest carries the SHA-1 hash of a data segment.
message HMACRequest {
  // The SHA-1 hash of the input data. SHA-1 hash is strictly 20 bytes.
  bytes sha1_hash = 1;
}

// HMACResponse carries the computed HMAC-SHA-1 signature.
message HMACResponse {
  // The HMAC-SHA-1 signature of the SHA-1 hash.
  bytes hmac = 1;
}
```

Messages

```proto
syntax = "proto3";

package pb;

option go_package = "kumari.net/blinded_unique_identifers/pb";

// IdentifierService defines a service for computing HMAC signatures of client hashes.
service IdentifierService {
  // GetHMAC receives a SHA-1 hash and returns its HMAC-SHA-1 signature.
  rpc GetHMAC(HMACRequest) returns (HMACResponse);
}

// HMACRequest carries the SHA-1 hash of a data segment.
message HMACRequest {
  // The SHA-1 hash of the input data. SHA-1 hash is strictly 20 bytes.
  bytes sha1_hash = 1;
}

// HMACResponse carries the computed HMAC-SHA-1 signature.
message HMACResponse {
  // The HMAC-SHA-1 signature of the SHA-1 hash.
  bytes hmac = 1;
}
```

Server Code

```go
// GetHMAC handles the client RPC request. It validates the SHA-1 input and returns the computed HMAC signature.
func (s *serverImpl) GetHMAC(ctx context.Context, req *pb.HMACRequest) (*pb.HMACResponse, error) {
        inputHash := req.GetSha1Hash()

        // Extract client IP address for contextual logging
        clientIP := "unknown"
        if p, ok := peer.FromContext(ctx); ok {
                clientIP = p.Addr.String()
        }

        // Direct HMAC-SHA-1 allocation
        mac := hmac.New(sha1.New, s.hmacKey)
        _, _ = mac.Write(inputHash) // Write never returns an error for hash.Hash interface
        signature := mac.Sum(nil)

        // Increment successful hashes statistics count
        statsTracker.AddHash()

        return &pb.HMACResponse{
                Hmac: signature,
        }, nil
}
```

Questions...

Backup...

- But you can't just publish these in WHOIS/RDAP/etc...
  - I kinda think you can, but if not, they could stil be shared during incidents

- What if registrars abuse this and perform dictionary probes?!
  - Ways to mitigate this (e.g redeemable tokens), but they add complexity, happy to discuss.