Privacy Aware Sharing of IOCs in MISP

Master thesis presentation for obtaining the Master's degree in Computer Science and Engineering

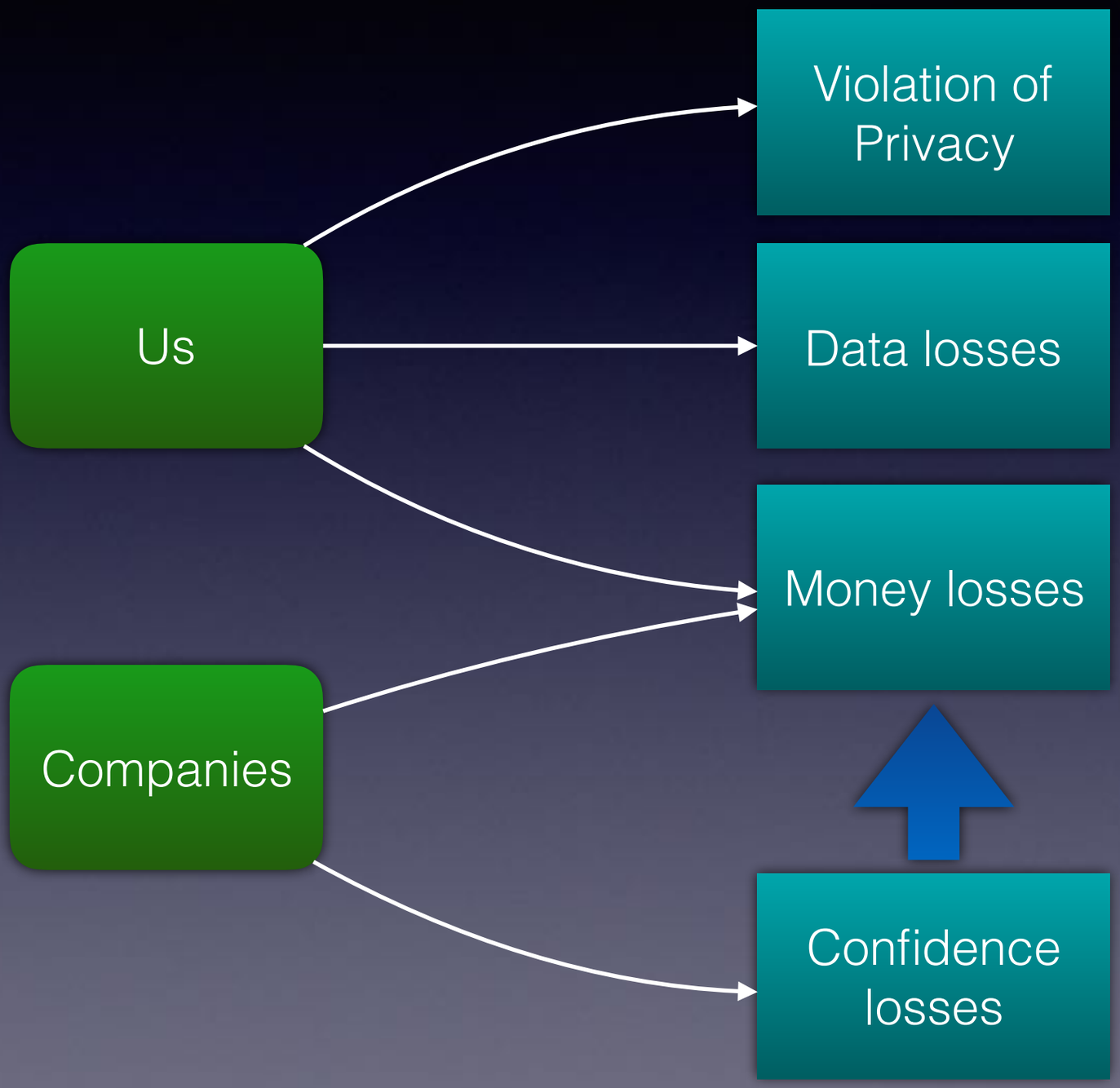
By Charles Jacquet, June 27

Supervisor: Ramin SADRE
With help from Conostix and CIRCL

Outline

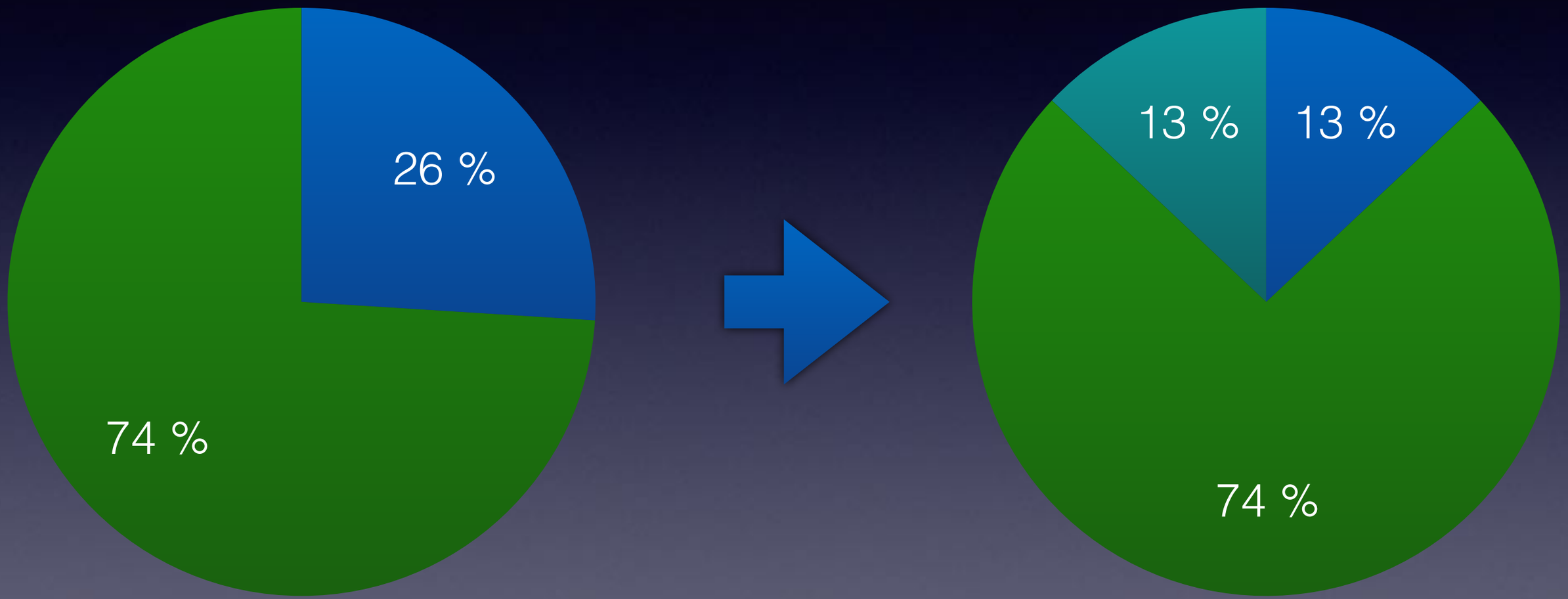
- Computer security
- Information Sharing
- Challenges
- Hashstore
- Possible solution
- Results

Computer Security



Computer Security

Ponemon Institute:



- 26% chance of data breach of more than 10 000 records in the next two years.
- 50% of data breaches are due to malicious attacks in France

Information Sharing

Clues of the attacks

Analyses of the attacks

Guidelines

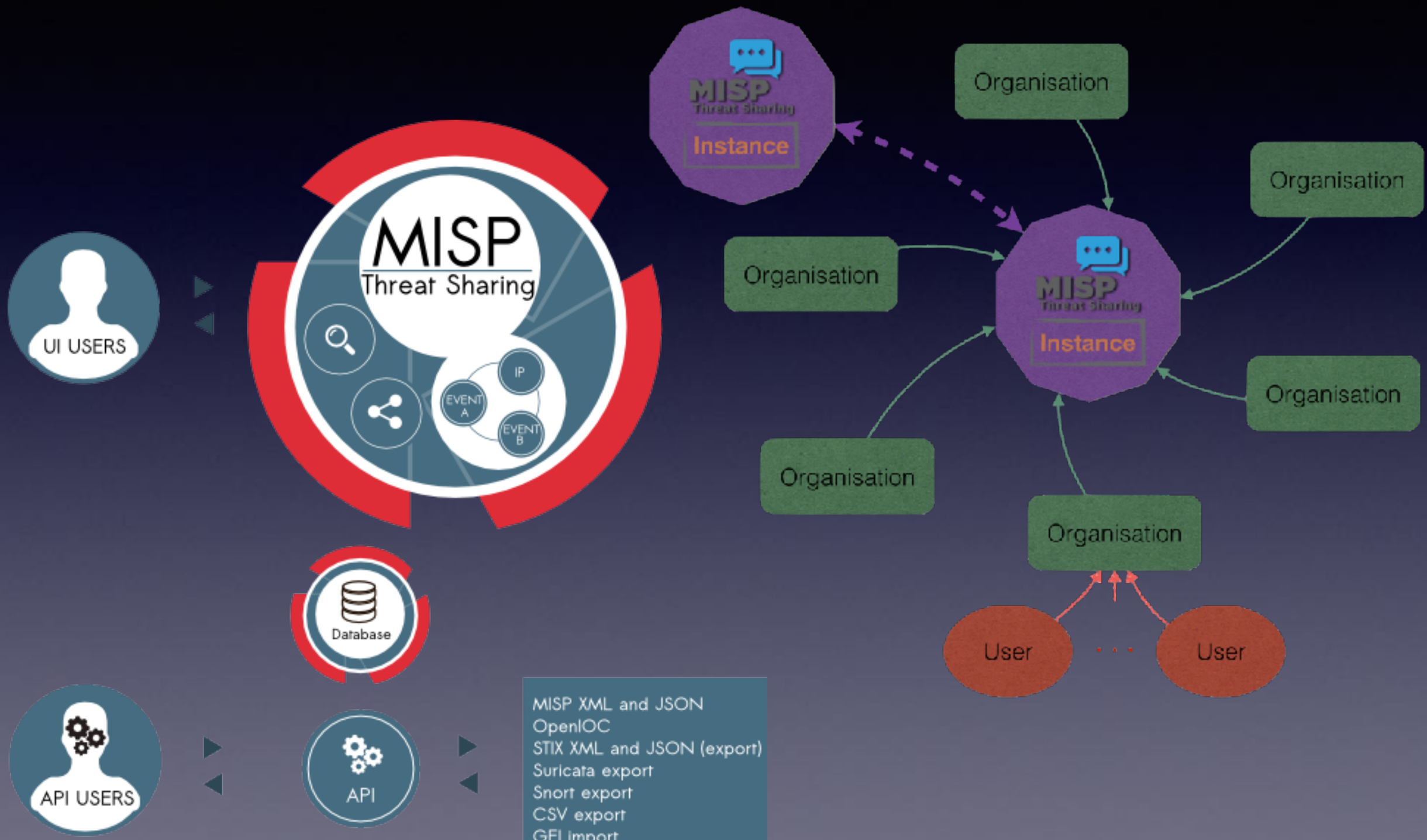
= Indicator Of Compromise (IOC)

- Urls
- IPs
- Malware samples
- Hashes
- …

Standards

Platforms

Malware Information Sharing Platform (MISP)



Challenges

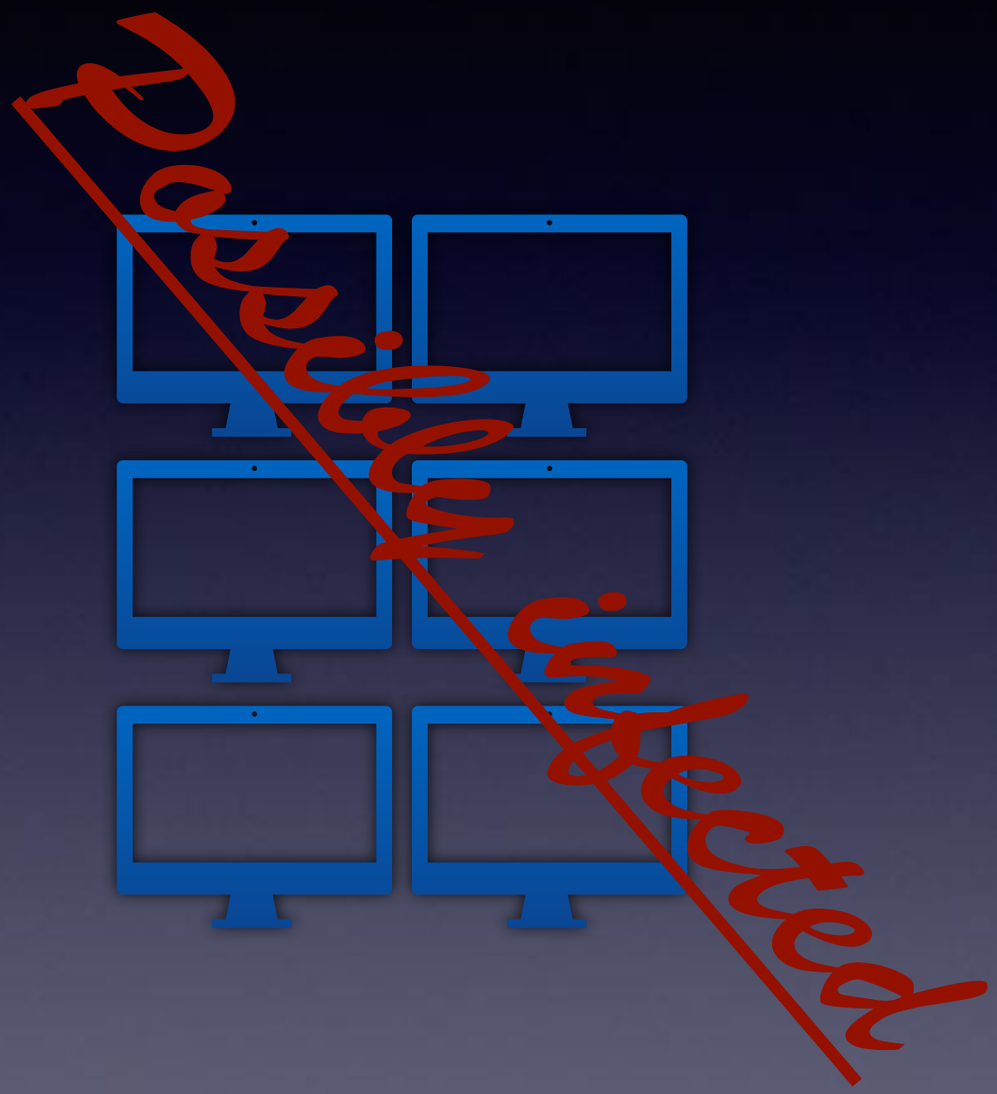
Confidentiality

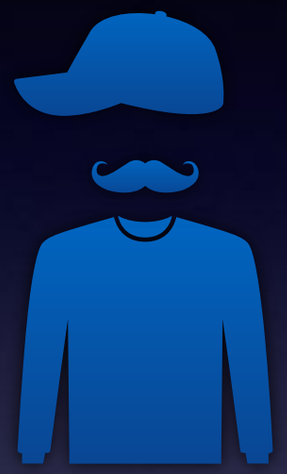
Privacy

Relies on
TRUST
In communities

Small sharing
communities

Challenges



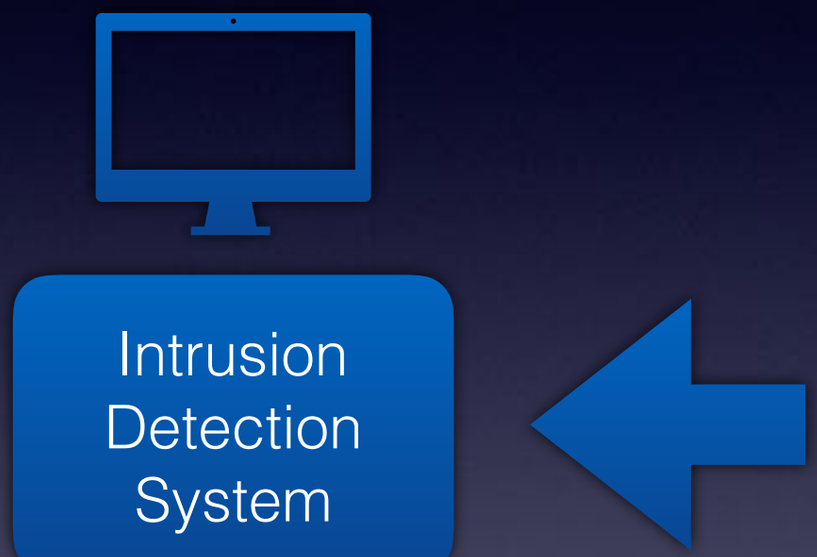


Forensic team

Should they connect their MISP instance of confidential data?

Should they bring all IOCs from MISP with them?

Challenges



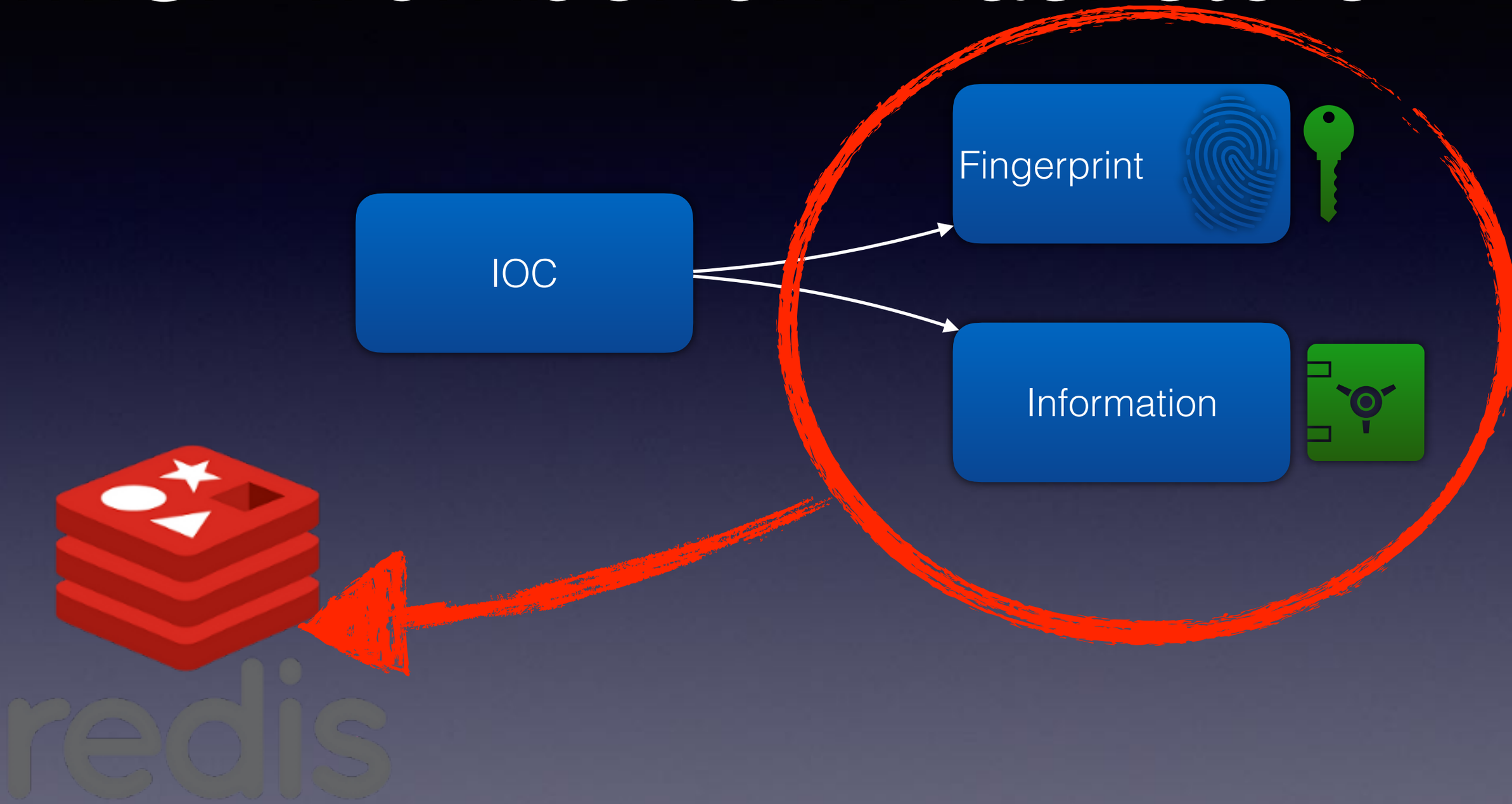
IDS rules generated with information available for our organisation in MISP

Could we create rules with information not accessible from our organization but that the organization agrees to share that it is an IOCs?

Need for new way of sharing

- Increasing the size of sharing communities
- Protecting Privacy and Confidentiality
- Make data sharable
- Make data usable even in compromised environnements

MISP workbench: Hashstore



Small data

IPv4 addresses

4 billion possibilities

109.88.52.167

3.7 billion possibilities

What about bruteforce attacks?

MISP workbench: Hashstore

508 388 get/s

IPv4 addresses

109.88.52.167

3.7 billion possibilities

A solution

van de Kamp, T., Peter, A., Everts, M. H., & Jonker, W. (2016, October). Private Sharing of IOCs and Sightings. In Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security (pp. 35-38). ACM.

The idea step by step

IOC:

- Url: www.ioc.com
- IP: 109.88.52.167
- Port number: 80
- UUID: 12345

- The type says how the rule is created
- The message says the user how to retrieve information from MISP

Rule:

- Type: url||ip||port
- Value: www.ioc.com ||109.88.52.167||80
- Message: uuid: 12345

The idea step by step

Rule:

- Type: url||ip||port
- Value: www.ioc.com ||109.88.52.167||80
- Message: uuid:12345

Hash the value

Rule:

- Type: url||ip||port
- ValueHash: 852da56165f1cea48fd8d09ddc081af7e1734be8
- Message: uuid:12345

The idea step by step

Rule:

- Type: url||ip||port
- ValueHash: 852da56165f1cea48fd8d09ddc081af7e1734be8
- Message: uuid:12345

Protect the message

Rule:

- Type: url||ip||port
- ValueHash: 852da56165f1cea48fd8d09ddc081af7e1734be8
- EValueHash(Message): [illegible]

The idea step by step

Rule:

- Type: url||ip||port
- ValueHash: 852da56165f1cea48fd8d09ddc081af7e1734be8
- $E_{ValueHash}$(Message): JÇyZ1šóv“ »M=Vj

Delete Hash

Rule:

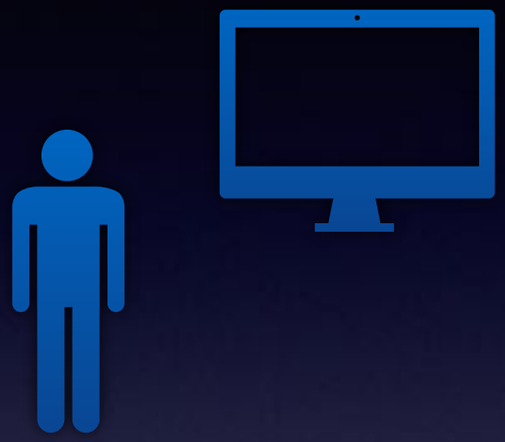
- Type: url||ip||port
- $E_{ValueHash}$(Message): JÇyZ1šóv“ »M=Vj

Rule:

- Type: url||ip-dst||port
- IV (base64): dqvKZS9ZIyJzDto312dmeg==
- Salt (base64): 9IK116Zri1I3R4/X3XiqHkVK2nnEB2GCBEH1cHO52pY=
- $E_{ValueHash}$(Message): Oçˆ‹žÜ'KTŽÓ4™ltg3Çš"’¿A*÷ /i:F

- Use salt for the Key Derivation Function (instead of HASH)
- Use IV for the AES encryption
- Use Identifier of the user inside the KDF

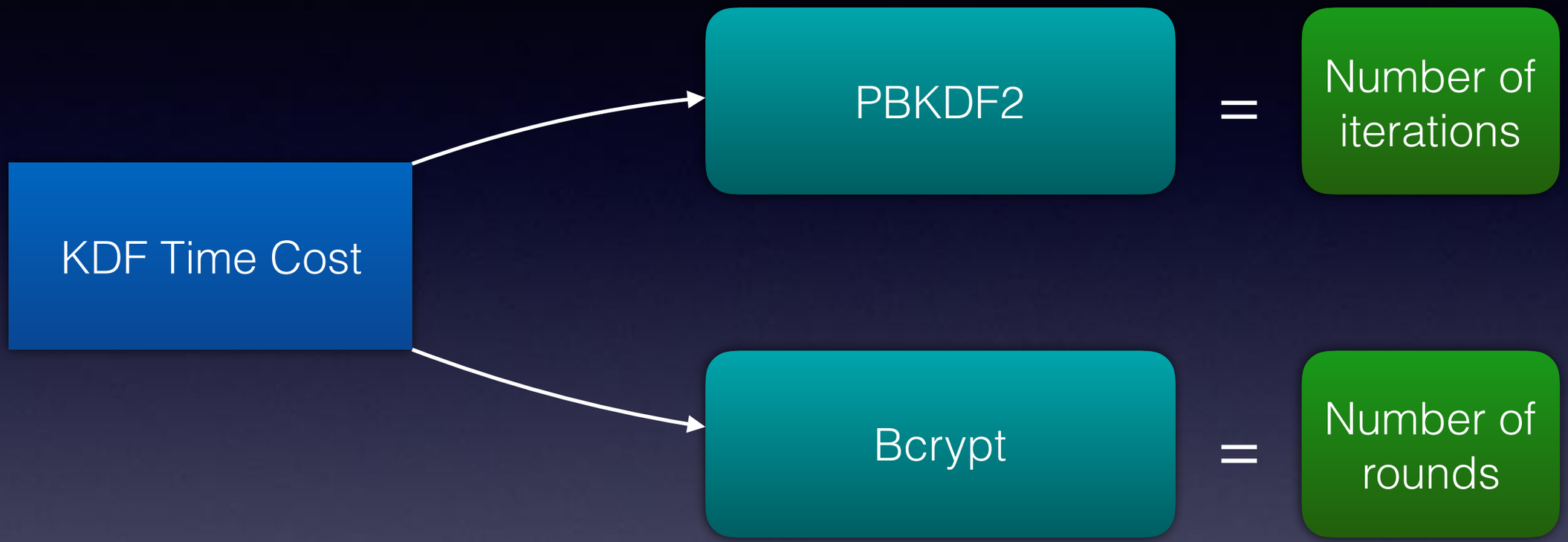
Check rules



Url= www.ReallyBadURL.com
Port= 80

```
Password = ‘www.ReallyBadURL.comll80’  + MISPToken
For each rule in rules like url||port do
                    Pass = KDF(rule.salt, Password)
                    Match = AES.decrypt(rule.message, rule.IV, Pass)
                    If Match not null do
                                Print(Match.identifer)
```

Parametrizable



There is a minimum cost by algorithm in order to generate a random looking key

More parametrizable

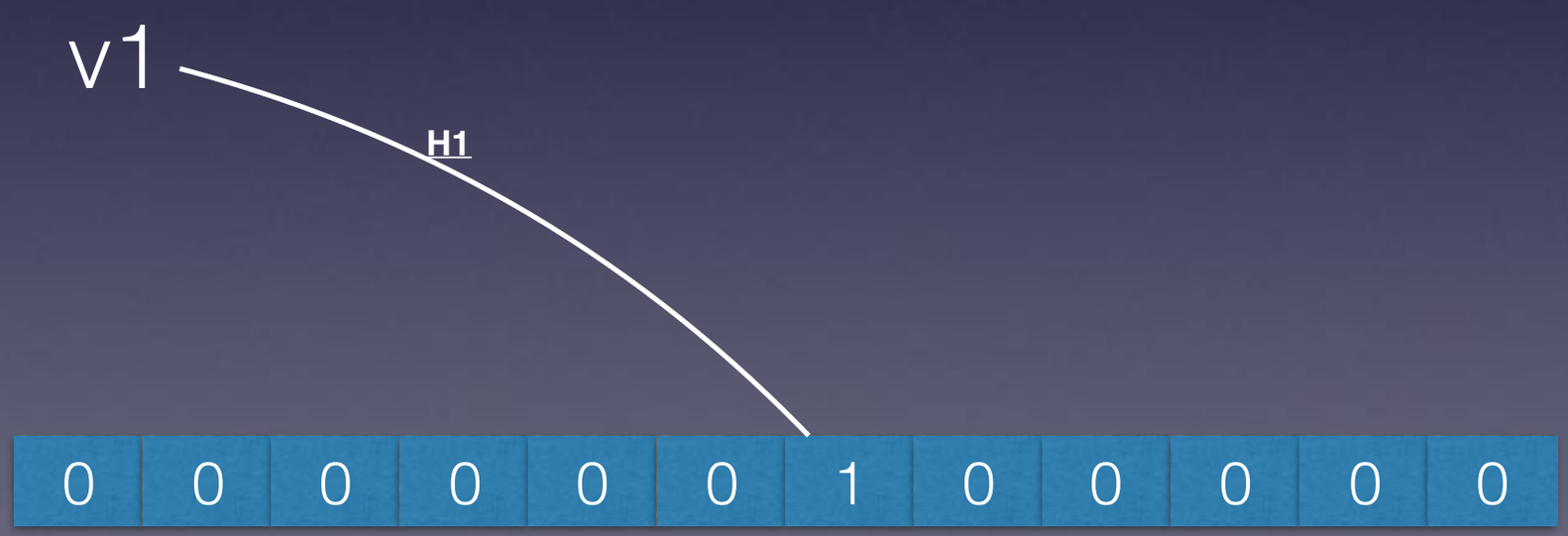
Bloom filters = Bloom filter is a space efficient probabilistic data structure used to efficiently test the membership of specific values

More parametrizable

Bloom filters = Bloom filter is a space efficient probabilistic data structure used to efficiently test the membership of specific values

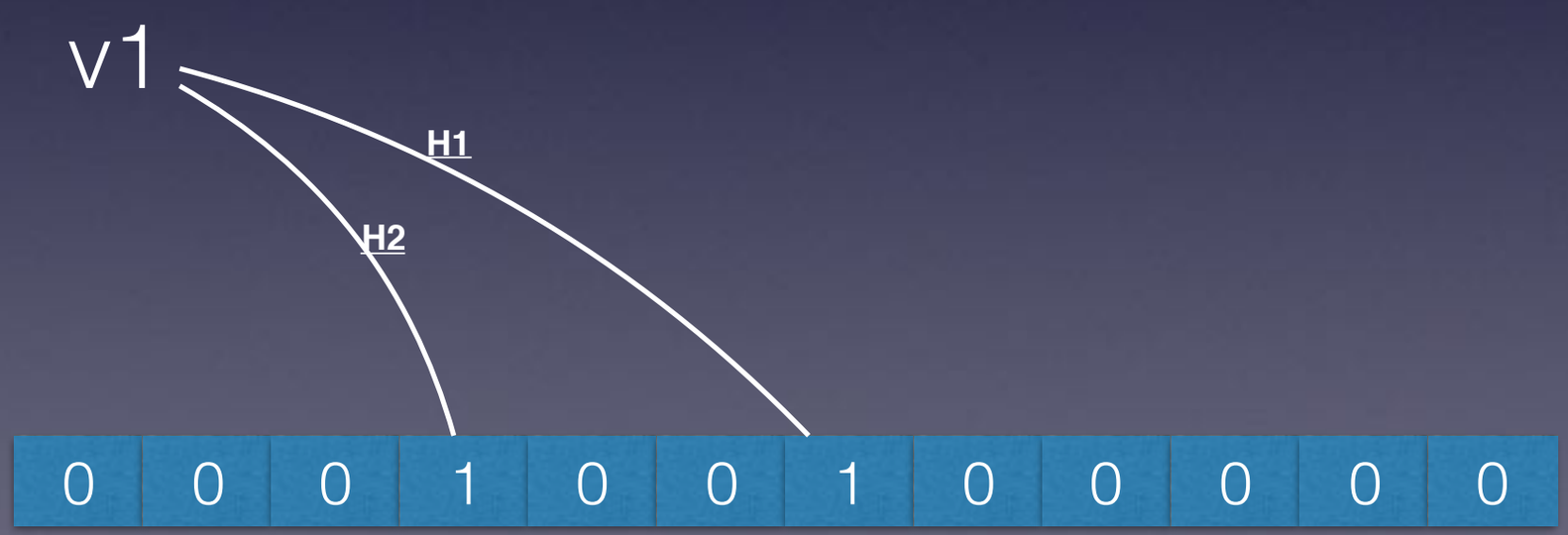
More parametrizable

Bloom filters = Bloom filter is a space efficient probabilistic data structure used to efficiently test the membership of specific values



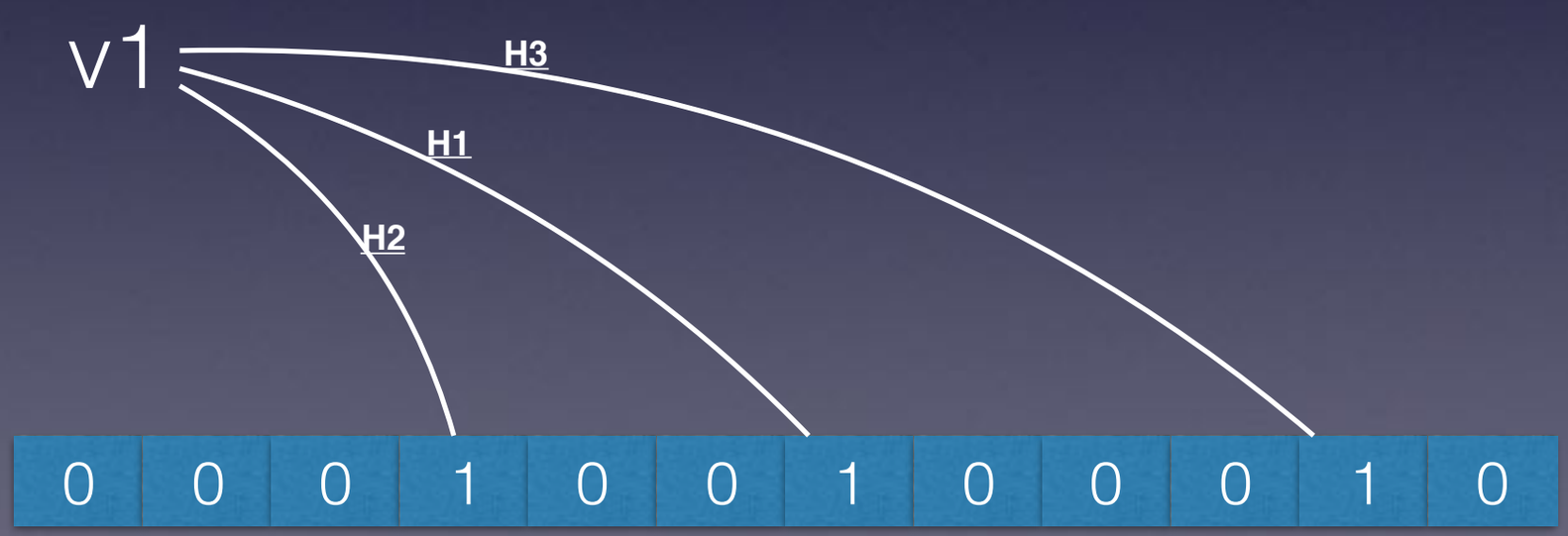
More parametrizable

Bloom filters = Bloom filter is a space efficient probabilistic data structure used to efficiently test the membership of specific values



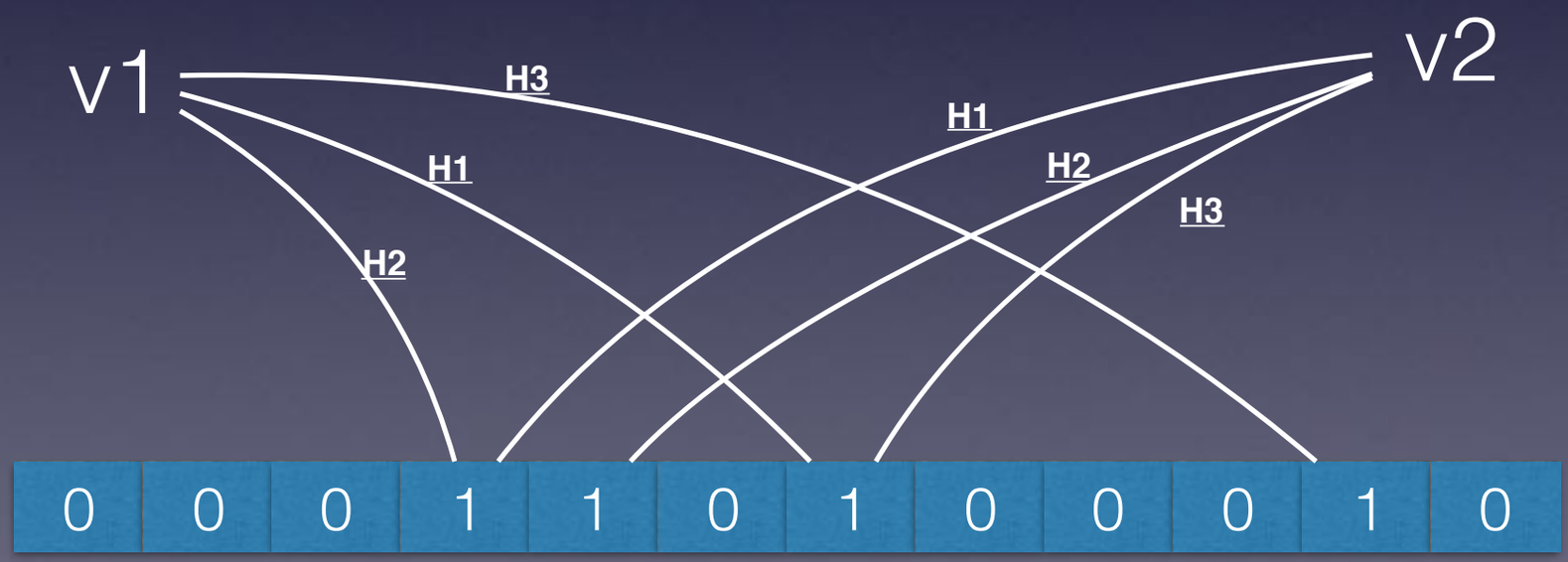
More parametrizable

Bloom filters = Bloom filter is a space efficient probabilistic data structure used to efficiently test the membership of specific values



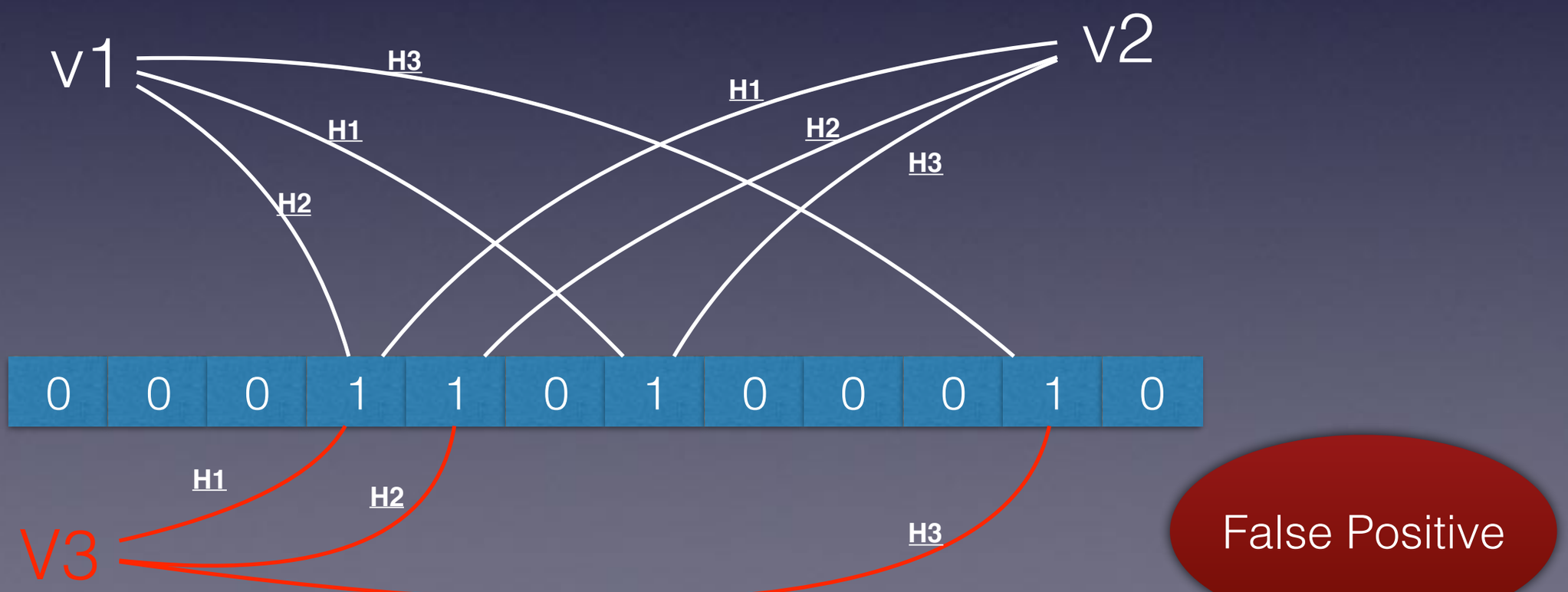
More parametrizable

Bloom filters = Bloom filter is a space efficient probabilistic data structure used to efficiently test the membership of specific values



More parametrizable

Bloom filters = Bloom filter is a space efficient probabilistic data structure used to efficiently test the membership of specific values



False Positive

More parametrizable

```
Password = ‘www.ReallyBadURL.comll80’  + MISPToken
If Password is in Bloom Filter do
        For each rule in rules like url||port do
                Pass = KDF(rule.salt, Password)
                Match = AES.decrypt(rule.message, rule.IV, Pass)
                If Match not null do
                        Print(Match.identifer)
```

The false positive rate can also be used to parametrize the matching speed

Results

Key Derivation Function

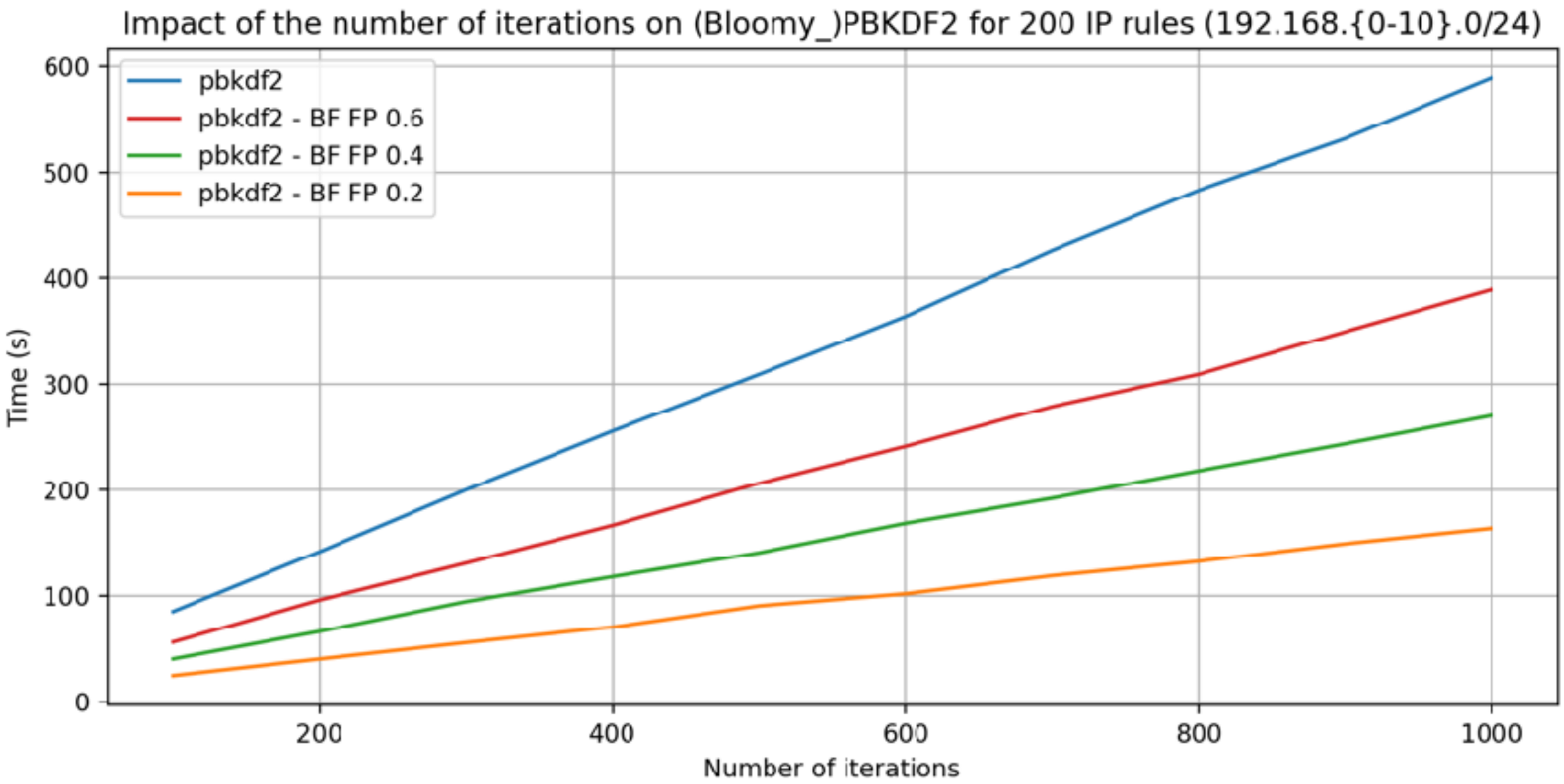
Protect the IOC value

Slow down a brute force attack

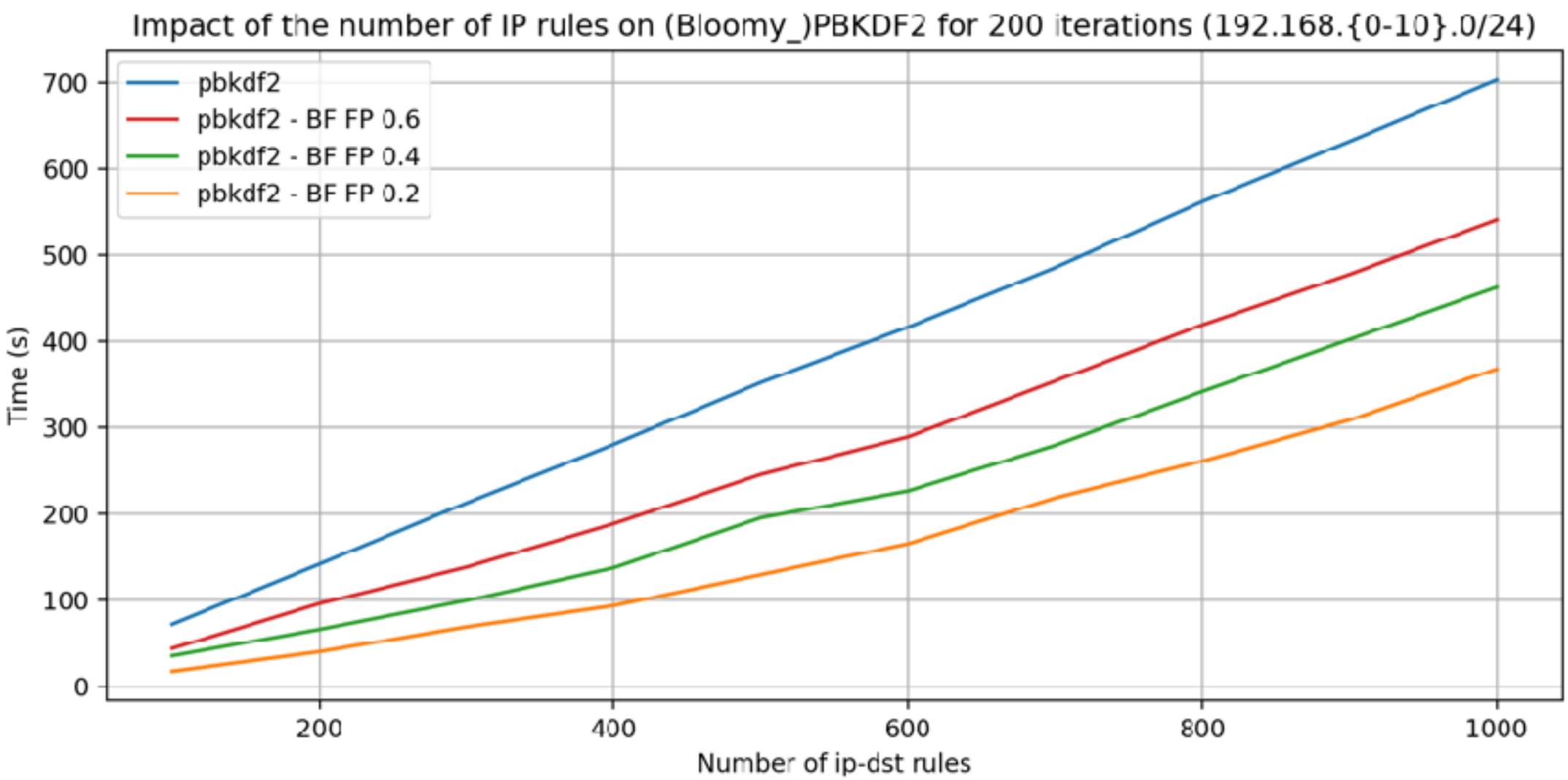
Bloom Filter

Increase matching speed if too slow

Results


Impact of the number of iterations on (Bloomy_)PBKDF2 for 200 IP rules (192.168.{0-10}.0/24)
pbkdf2
pbkdf2 - BF FP 0.6
pbkdf2 - BF FP 0.4
pbkdf2 - BF FP 0.2
Time (s)
0
100
200
300
400
500
600
200
400
600
800
1000
Number of iterations

Results


Impact of the number of IP rules on (Bloomy_)PBKDF2 for 200 iterations (192.168.{0-10}.0/24)
pbkdf2
pbkdf2 - BF FP 0.6
pbkdf2 - BF FP 0.4
pbkdf2 - BF FP 0.2
Time (s)
0
100
200
300
400
500
600
700
200
400
600
800
1000
Number of ip-dst rules

Results

- KDFTime = Time for 1 iteration on a value
- nValueTested = [nRules + (nValues-nRules)•FPrate]
- BruteforceTime = nValueTested•nRules•KDFTime•KDFCost

Results

How to choose the cost and FP rate?

BruteforceTime $\geq$ User time specification for bruteforce

Conclusion

- Increasing the size of sharing communities

Conclusion

- Increasing the size of sharing communities
- Protecting Privacy and Confidentiality

Conclusion

- Increasing the size of sharing communities
- Protecting Privacy and Confidentiality
- Make data sharable

Conclusion

- Increasing the size of sharing communities
- Protecting Privacy and Confidentiality
- Make data sharable
- Make data usable even in compromised environnements

But Not Perfect !

An attacker with big computation power

Will manage to succeed a bruteforce attack as it is parallelisable and linear!

Further Work

Avoid parallelisation

Allow anonymous IOC sighting reporting

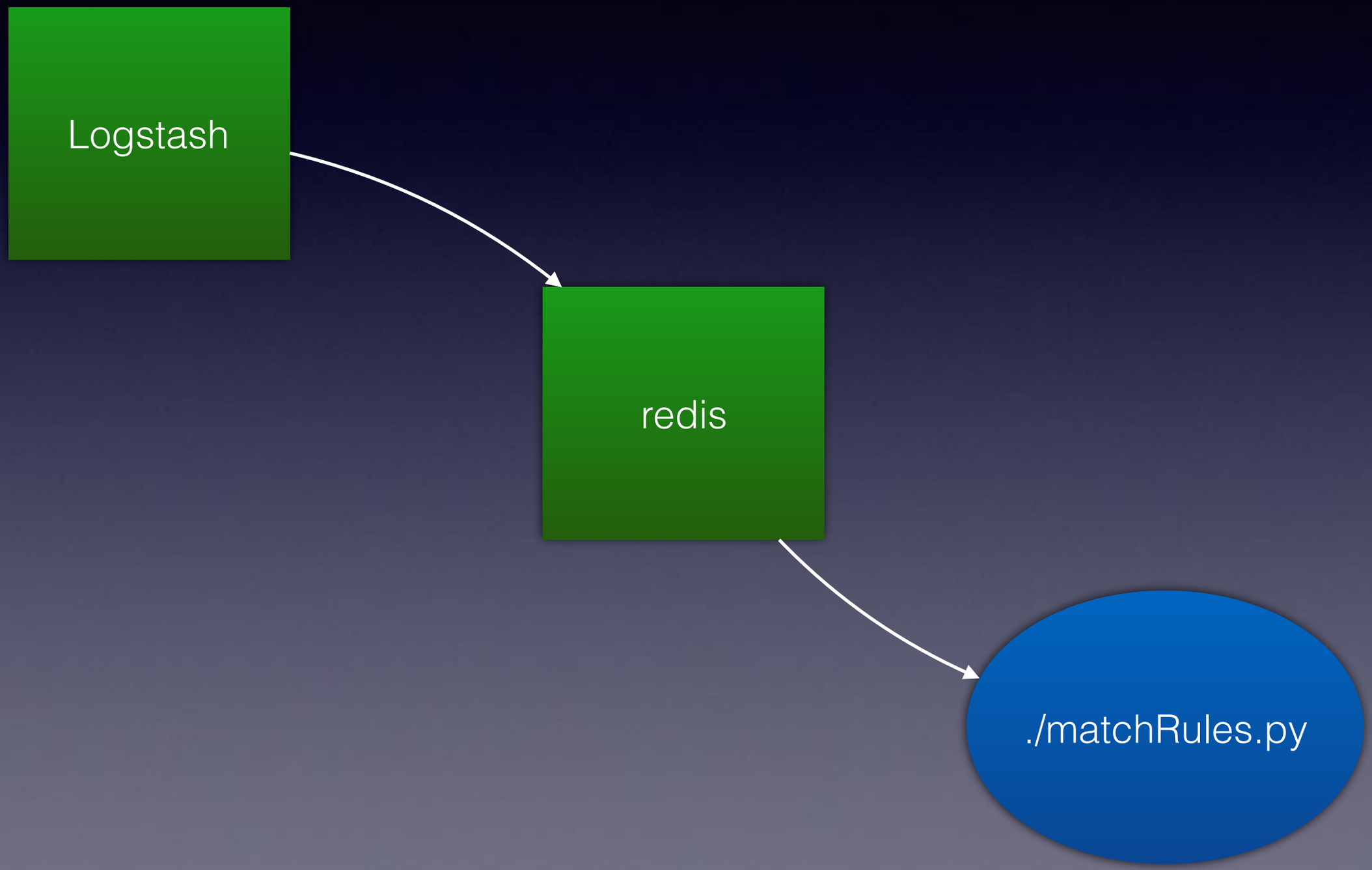
Explore new techniques

Thank you for your attention

Conclusion

- Increasing the size of sharing communities
- Protecting Privacy and Confidentiality
- Make data sharable
- Make data usable even in compromised environnements

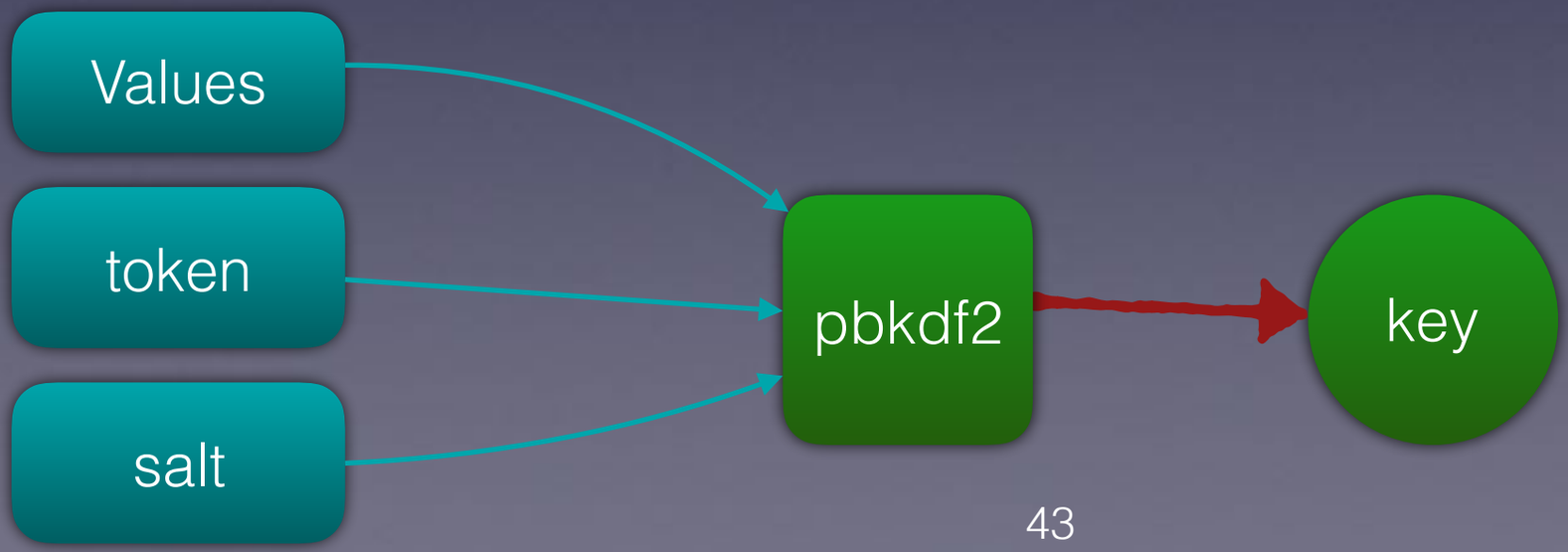
Pipe to read logs



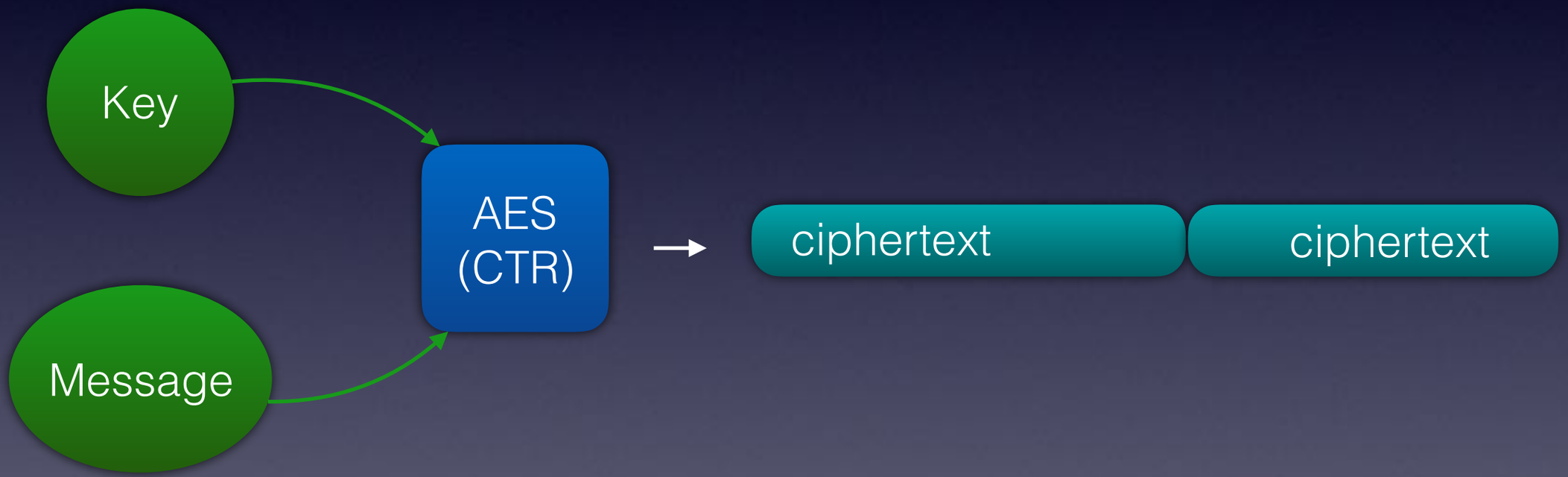
Pbkdf2 vs HKDF

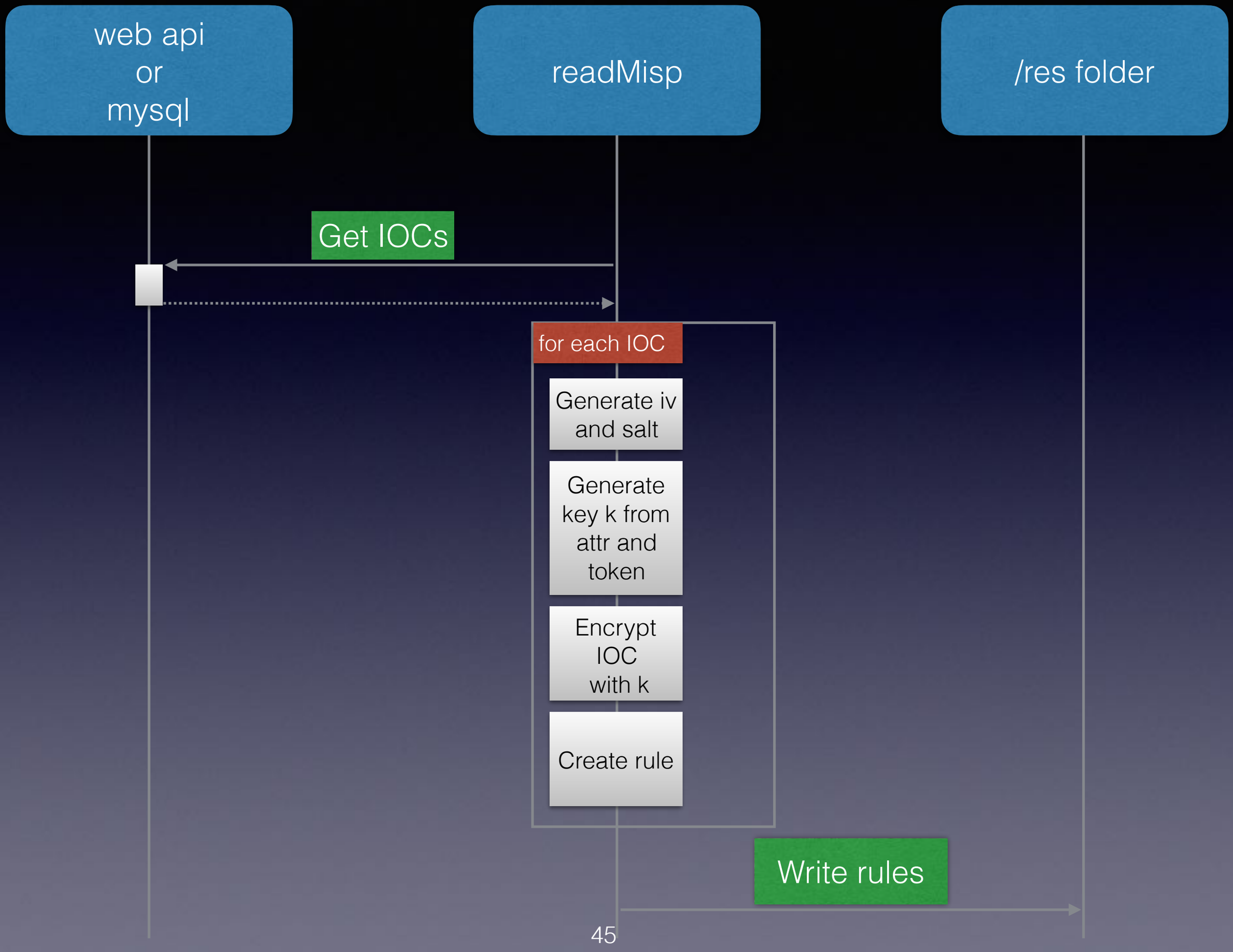
Slow down brute force thanks to iterations

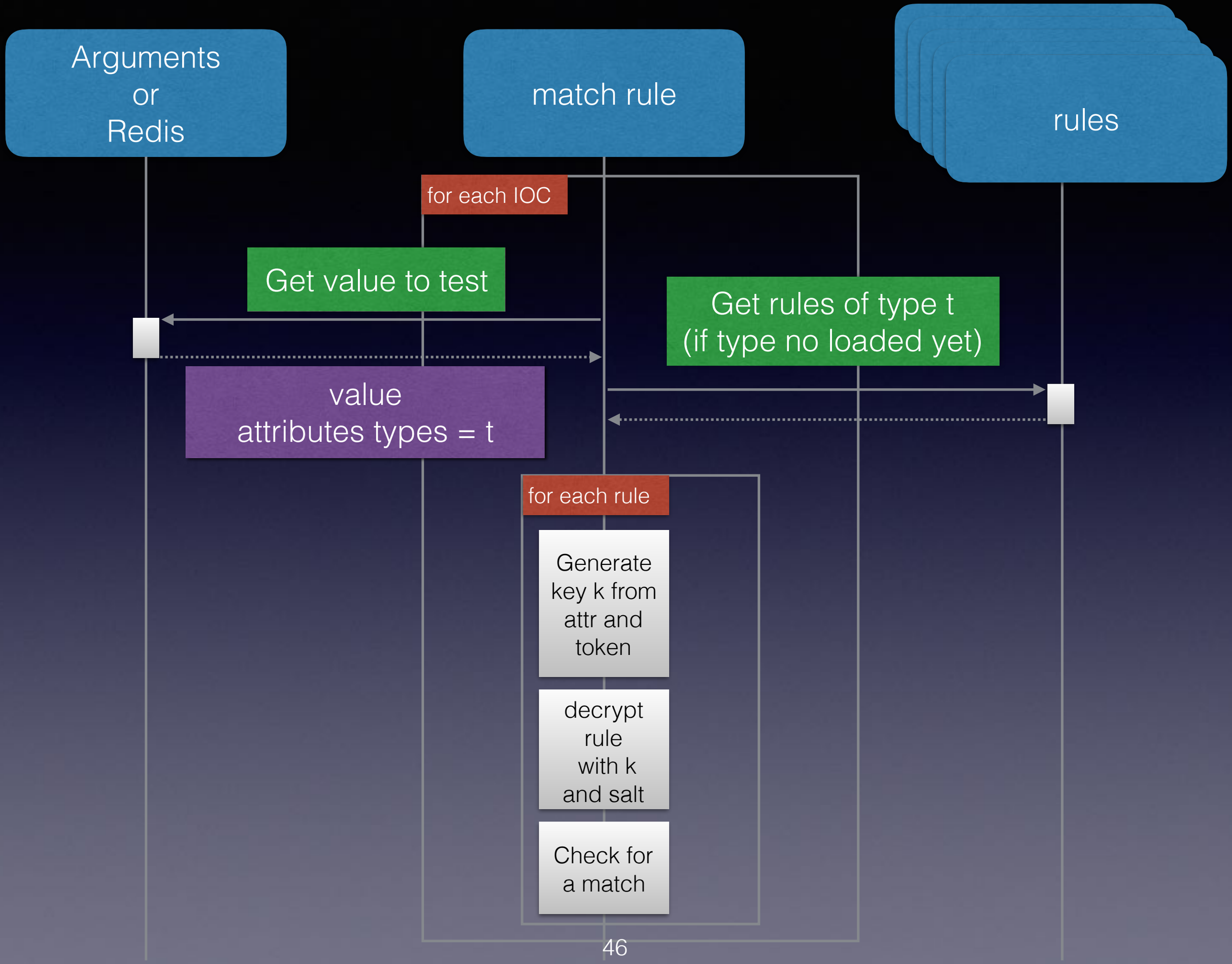
Designed to be « random looking » directly

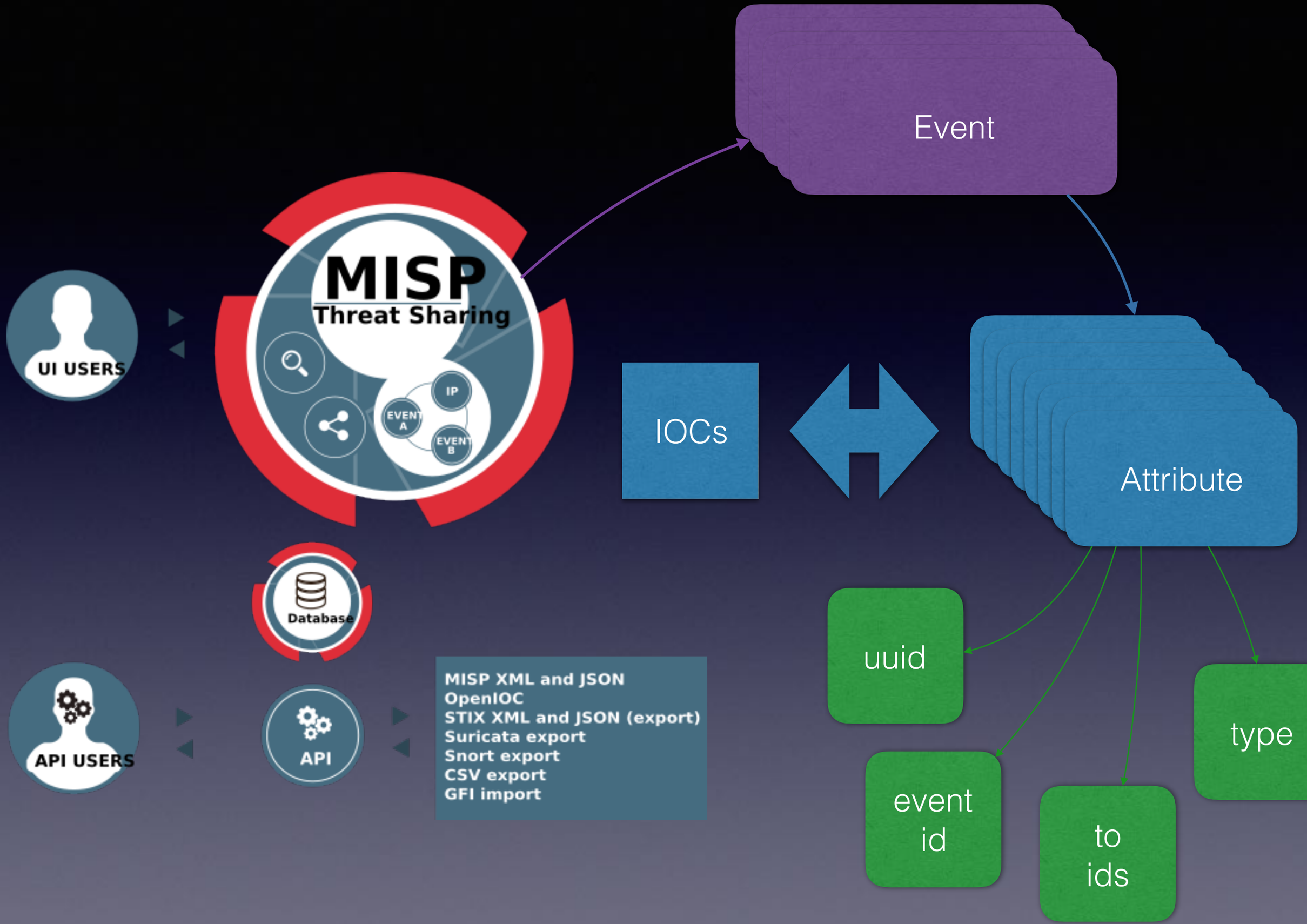


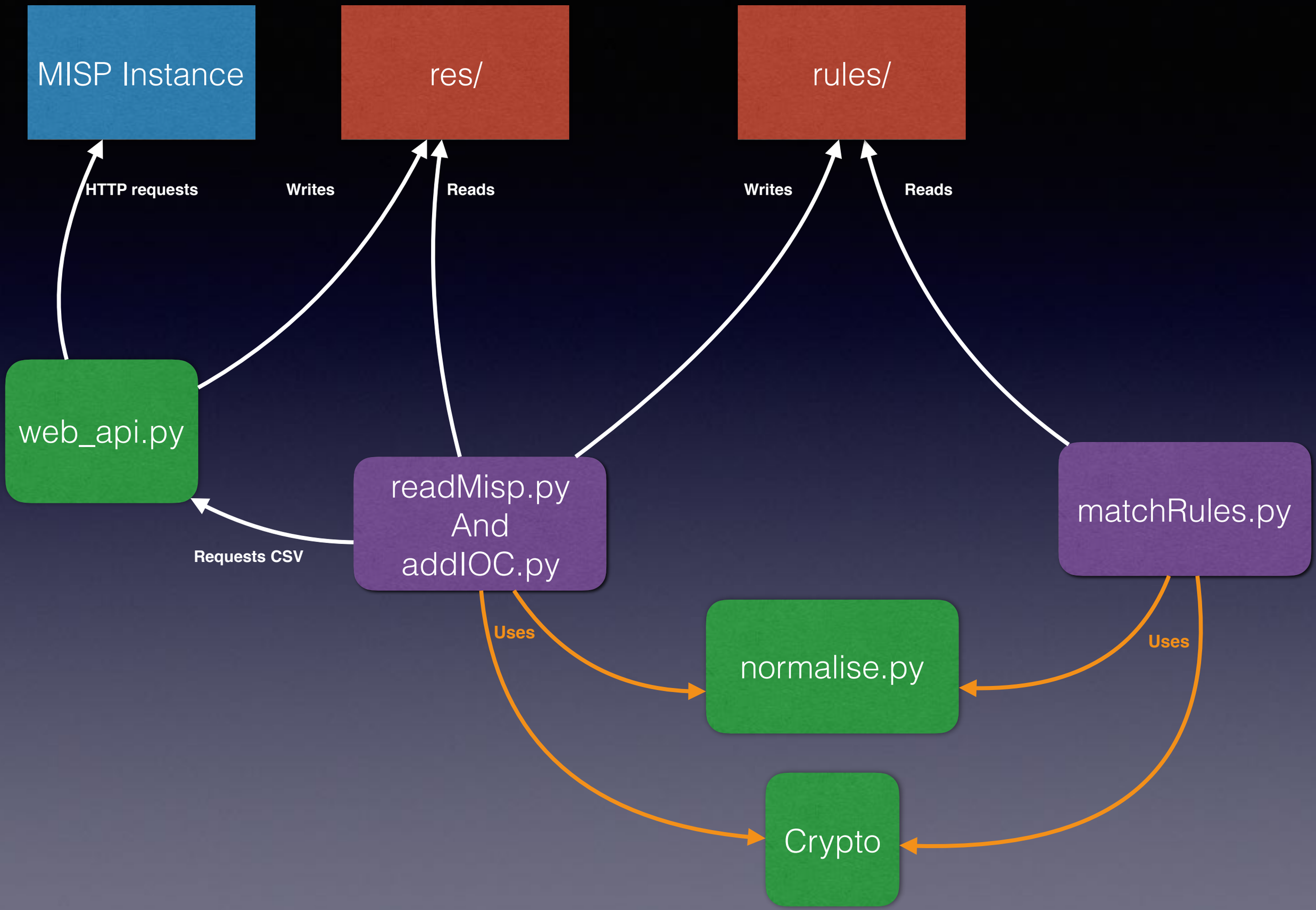
Encryption

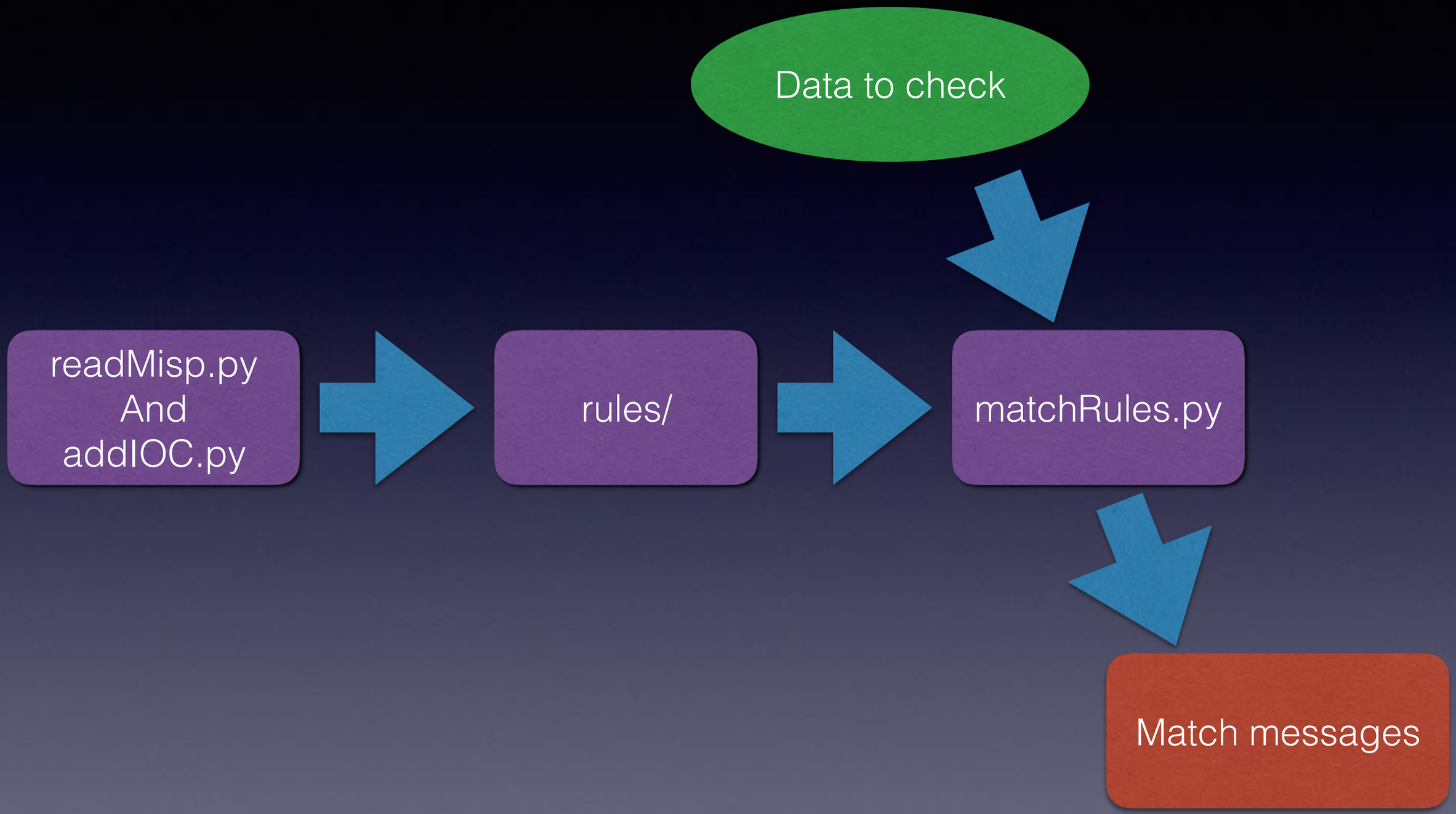



web api
or
mysql
readMisp
/res folder
Get IOCs
for each IOC
Generate iv
and salt
Generate
key k from
attr and
token
Encrypt
IOC
with k
Create rule
Write rules


Arguments
or
Redis
match rule
rules
for each IOC
Get value to test
Get rules of type t
(if type no loaded yet)
value
attributes types = t
for each rule
Generate
key k from
attr and
token
decrypt
rule
with k
and salt
Check for
a match


Event
MISP
Threat Sharing
UI USERS
IP
EVENT A
EVENT B
IOCs
Attribute
uuid
event id
to ids
type
Database
API USERS
API
MISP XML and JSON
OpenIOC
STIX XML and JSON (export)
Suricata export
Snort export
CSV export
GFI import


MISP Instance
res/
rules/
HTTP requests
Writes
Reads
Writes
Reads
web_api.py
readMisp.py
And
addIOC.py
matchRules.py
Requests CSV
Uses
Uses
normalise.py
Crypto


Data to check
readMisp.py
And
addIOC.py
rules/
matchRules.py
Match messages

Configuration

```
[misp]
token = put_your_token
email = put_your_email
url = put_web_api_address

[rules]
location = ../rules
cryptoModule = pbkdf2
message = uuid event_id date

[bloomy]
fp_rate = 0.3

[pbkdf2]
iterations = 1000
ipiterations = 1000
hash_name = sha256
dklen = 32
```

PBKDF2 IP rules

Salt = 9IK116Zri1I3R4/X3XiqHkVK2nnEB2GCBEH1cHO52pY=
attributes = ip-dst
nonce = dqvKZS9ZIyJzDto312dmeg==
ciphertext-check = jyf4iOTXuT6Wm2KnpLQbnw==
ciphertext = I41HIEFNcEZZTjP0i0GTR1Qmfco3EJ9uWmNHYd88Lljg8L1b+hb2K/xxMQJhRQA=