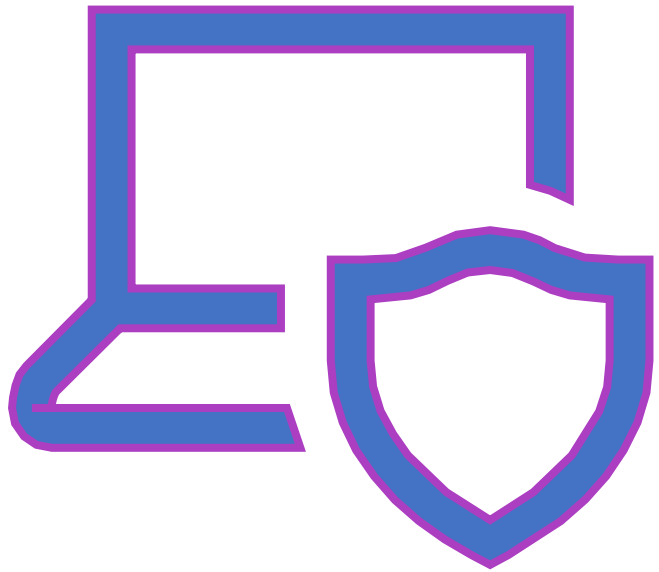
# SECURE CRYPTO CONFIG (SCC)

@IETF 109 / CRYPTO FORUM RESEARCH GROUP

KAI MINDERMANN, M.SC.



# SECURE CRYPTO CONFIG IN A NUTSHELL

- Builds upon **COSE format**
- SCC Process publishes yearly a **selected list of secure COSE Algorithm/-Configuration** Ids
- Cryptography Software libraries provide a **simplified interface** for **common cryptography operations** by using COSE as output format and using SCC configurations to allow both **automatic/default and easier manual selection of algorithms and their parameters.**

# PROBLEMS OF CRYPTOGRAPHY ALGORITHM INTERFACES/APIS

# PROBLEMS OF CRYPTOGRAPHY ALGORITHMS INTERFACES/APIS

Choosing Secure Parameters/-Combinations is really difficult

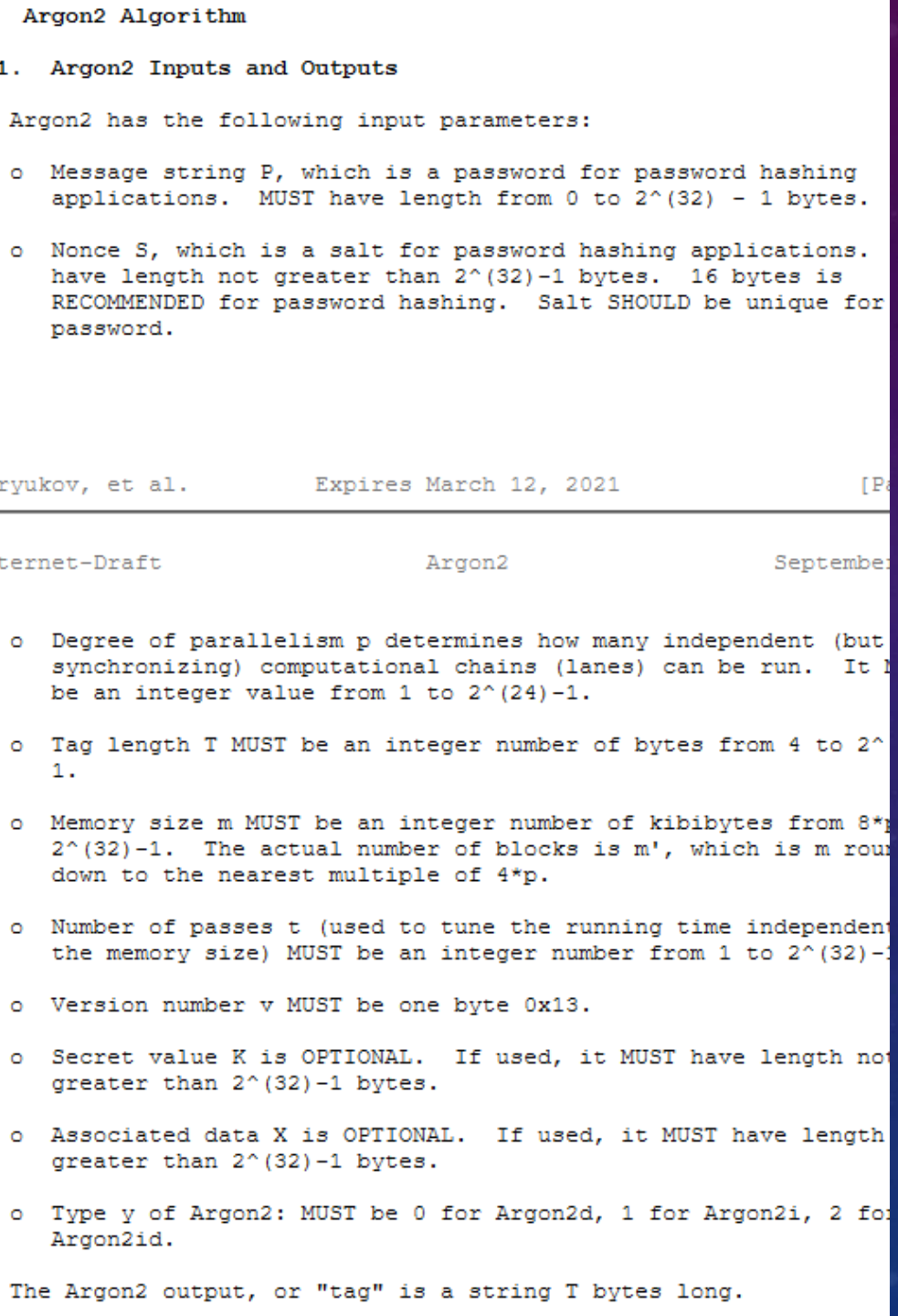
Establishing Secure Defaults is difficult to make future proof

It‘s difficult to make cryptography software libraries misuse resitant

# EXAMPLE 1: AES

1. What Key length?
2. Which block mode?
3. Which or at all Padding Algorithm?
4. Tag length?
5. Nonce length?

6. How to change these settings in the future?



Argon2 Algorithm

1. Argon2 Inputs and Outputs

Argon2 has the following input parameters:

o Message string P, which is a password for password hashing applications. MUST have length from 0 to 2^(32) - 1 bytes.

o Nonce S, which is a salt for password hashing applications. have length not greater than 2^(32)-1 bytes. 16 bytes is RECOMMENDED for password hashing. Salt SHOULD be unique for password.

ryukov, et al. Expires March 12, 2021 [P

ternet-Draft Argon2 Septembe

o Degree of parallelism p determines how many independent (but synchronizing) computational chains (lanes) can be run. It be an integer value from 1 to 2^(24)-1.

o Tag length T MUST be an integer number of bytes from 4 to 2^ 1.

o Memory size m MUST be an integer number of kibibytes from 8*p 2^(32)-1. The actual number of blocks is m', which is m rou down to the nearest multiple of 4*p.

o Number of passes t (used to tune the running time independen the memory size) MUST be an integer number from 1 to 2^(32)-

o Version number v MUST be one byte 0x13.

o Secret value K is OPTIONAL. If used, it MUST have length no greater than 2^(32)-1 bytes.

o Associated data X is OPTIONAL. If used, it MUST have length greater than 2^(32)-1 bytes.

o Type y of Argon2: MUST be 0 for Argon2d, 1 for Argon2i, 2 fo Argon2id.

The Argon2 output, or "tag" is a string T bytes long.

# EXAMPLE 2: ARGON2

1. Format for input string
2. Length of nonce
3. Number of threads (parallelism)
4. Length of tag
5. Number of bytes (memory size)
6. Number of passes (iterations)
7. Type of Argon2?
8. …

- How do I know that I chose a secure combination? (Yes there is a guide in the standard, that's very nice, but still difficult in practice)
- It's great that we have the Argon2, yet lot of implementation choices

# DEVELOPER EXPECTATION VS ALGORITHM FLEXIBILITY

- Assumption: Developers are not security experts
- Expected parameters to use symmetric encryption:
  1. Key
  2. Plaintext
  3. NOTHING MORE
- Expected parameters to use hashing:
  1. Plaintext
  2. NOTHING MORE

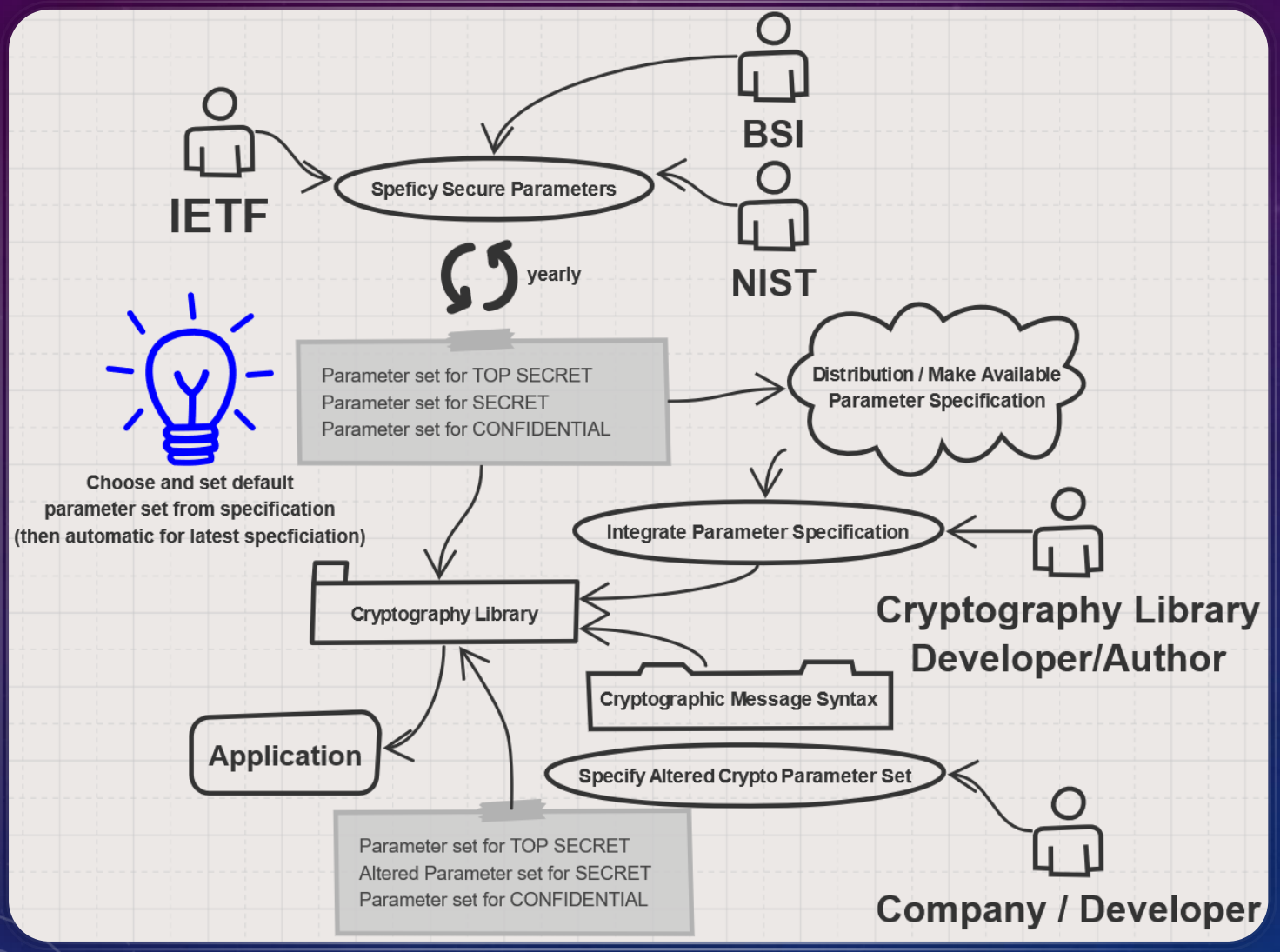
# SECURE CRYPTO CONFIG (SCC)

1. A **process** that is repeated every two years, where a new set of default configurations for standardized cryptography primitives is published in a standardized machine-readable format.
2. A Secure Crypto Config **Interface** that describes a common API to use cryptography primitives in software to be offered by standard libraries.
3. Using **COSE** to derive (and save) the parameters from output of cryptography primitives, otherwise future changes of the default configuration would change existing applications behavior.

https://datatracker.ietf.org/doc/draft-kaimindermann-securecryptoconfig/

https://github.com/secureCryptoConfig/secureCryptoConfig

# PROCESS




Mindermann, K. & Wagner, S., (2019). Towards a central, distributed and secure default cryptography parameter set. In: Kiefer, F. & Loebenberger, D. (Hrsg.), crypto day matters 30. Bonn: Gesellschaft für Informatik e.V. / FG KRYPTO. DOI: 10.18420/cdm-2019-30-25

(There I was still referring to Cryptographic Message Syntax (CMS) instead of COSE)

# CRYPTOGRAPHY BUILDING BLOCKS FOR THE SCC

**Configuration**

- Many per cryptography algorithm
- Distinct configuration
- Unique name for a configuration

**Format**

- Based on COSE (RFC 8152 https://tools.ietf.org/html/rfc8152)
- Identifier of algorithm (see above) and used parameters (e.g. lenghts and content like salt value)

**Output**

- Parsable for inverse operations (decryption/validation/…)
- Allows changes of default implementations

https://github.com/secureCryptoConfig/secureCryptoConfigInterface

# SECURE CRYPTO CONFIG – COMMON INTERFACE

## Package org.securecryptoconfig

The Secure Crypto Config Interface provides methods for the most common cryptographic use cases: **Symmetric encryption, Asymmetric encryption, hashing, password hashing and Signing.** The algorithms used for for the internal execution of the invoked use case is determined by the content of the Secure Crypto Config files. With the release of a new version of the Interface the files will be updated to use only currently secure algorithms and parameters. Therefore, it is necessary to update the SecureCryptoConfig library as soon as possible if a new version is provided to be able to be up-to-date with the current security standard. In this way the burden of making right choices for parameters and algorithms to implement secure code can be lifted from the user.

For implementing different use cases the most important methods can be found at `SecureCryptoConfig`. First create a instance of `SecureCryptoConfig` and invoke the corresponding method for the specific cryptographic use case that should be implemented.

One of the most importatnt use cases provided is the symmetric en/decryption. This can be realized with the Secure Crypto Config as follows:

```
byte[] plaintext = "Hello World!".getBytes(StandardCharsets.UTF_8);
SCCKey key = SCCKey.createKey(KeyUseCase.SymmetricEncryption);
SecureCryptoConfig scc = new SecureCryptoConfig();
// Encryption
SCCCiphertext ciphertext = scc.encryptSymmetric(key, plaintext);
// Decryption
PlaintextContainer plain = scc.decryptSymmetric(key, ciphertext);
```

# SCC SUMMARY

- The Secure Crypto Config (SCC) defines common use cases for cryptography operations
- The SCC offers standardized configuration sets for all required parameters of widely available cryptography algorithms for each common use case
- The SCC establishes a process to continuously provide updated configuration sets based on exsisting standardization processes of the IETF
- The SCC allows cryptography libraries to use standardized output formats of cryptography operations
- The SCC allows cryptography libraries to change their defaults based on the latest recommended configuation set (backwards compatbility ensured by output format with parameters)



https://datatracker.ietf.org/doc/draft-kaimindermann-securecryptoconfig/

# SOME OF THE OPEN QUESTIONS

- How to proceed with standardization?
- Is COSE the appropriate format to use for this
- Which is the ONE registry (IANA? Which one?) for cryptography algorithm suites (algorithm + parameter selection definition)
- What are the common cryptography use cases?
- What are appropriate Security Levels?
- What format should be used for the cryptographic signature of the published configurations?
- How can parameters that depend on the runtime environment be chosen automatically (e.g. memory/threads to be used for Argon2)?
- Key formats?

# HOW TO CONTRIBUTE?

- Provide feedback on the Draft via the CFRG Mailinglist.
- Contribute and provide feedback through issues for the Draft on GitHub: https://github.com/secureCryptoConfig/secureCryptoConfig
- Contribute, evaluate and provide feedback to the preliminary Java interface: https://github.com/secureCryptoConfig/secureCryptoConfigInterface