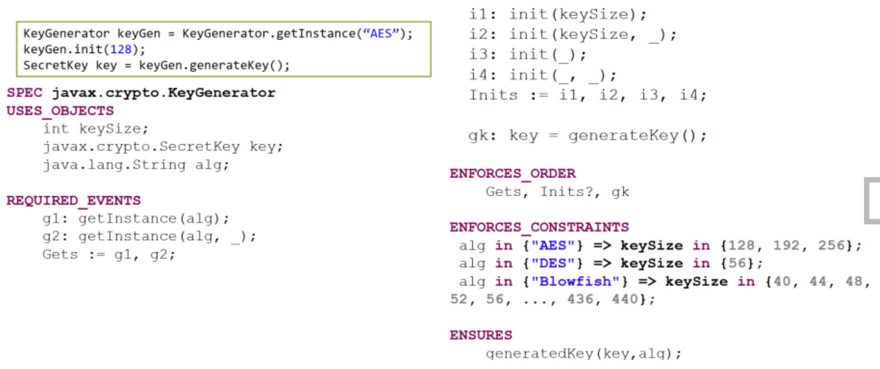
Mastering Variability in Crypto APIs

Speaker: Stefan Krüger

Crypto APIs

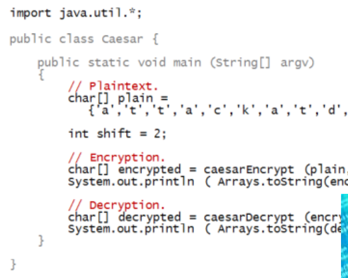
- Studies show: Hard to use
- Solution:



```
KeyGenerator keyGen = KeyGenerator.getInstance("AES");
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

```
SPEC javax.crypto.KeyGenerator
USES_OBJECTS
    int keySize;
    javax.crypto.SecretKey key;
    java.lang.String alg;

REQUIRED_EVENTS
    g1: getInstance(alg);
    g2: getInstance(alg, _);
    Gets := g1, g2;
```

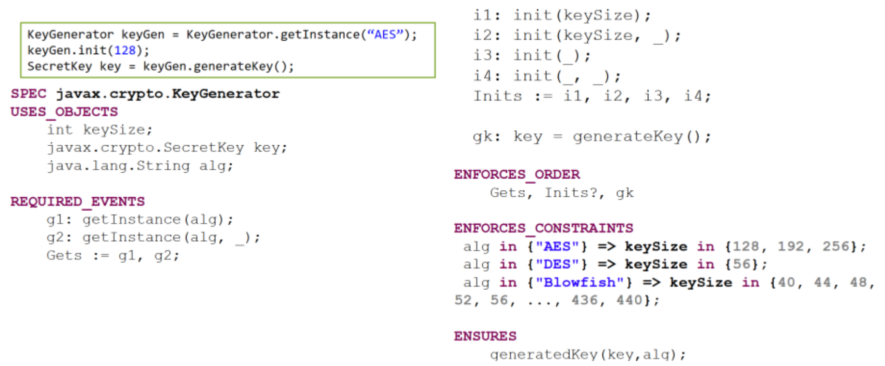
```
  i1: init(keySize);
  i2: init(keySize, _);
  i3: init(_);
  i4: init(_, _);
  Inits := i1, i2, i3, i4;

  gk: key = generateKey();

ENFORCES_ORDER
    Gets, Inits?, gk

ENFORCES_CONSTRAINTS
 alg in {"AES"} => keySize in {128, 192, 256};
 alg in {"DES"} => keySize in {56};
 alg in {"Blowfish"} => keySize in {40, 44, 48,
52, 56, ..., 436, 440};

ENSURES
    generatedKey(key,alg);
```



```
import java.util.*;

public class Caesar {

    public static void main (String[] argv)
    {
        // Plaintext.
        char[] plain =
           {'a','t','t','a','c','k','a','t','d','a','w','n'};

        int shift = 2;

        // Encryption.
        char[] encrypted = caesarEncrypt (plain, shift);
        System.out.println ( Arrays.toString(encrypted) );

        // Decryption.
        char[] decrypted = caesarDecrypt (encr
        System.out.println ( Arrays.toString(de
    }

}
```

Crypto APIs

- Studies show: Hard to use
- Solution:



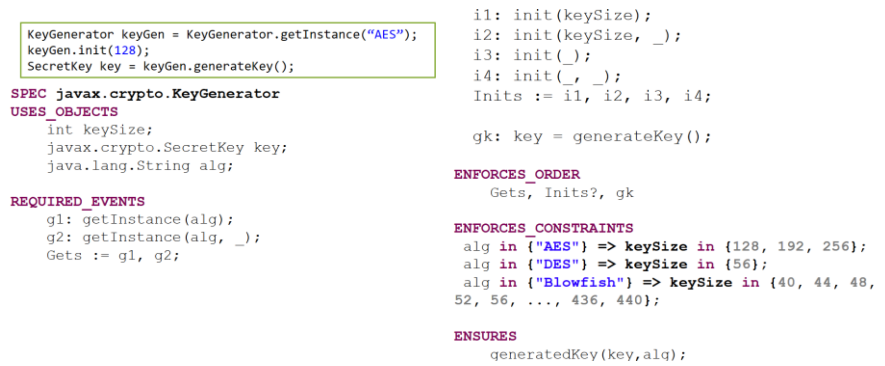
```
KeyGenerator keyGen = KeyGenerator.getInstance("AES");
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

```
SPEC javax.crypto.KeyGenerator
USES_OBJECTS
    int keySize;
    javax.crypto.SecretKey key;
    java.lang.String alg;

REQUIRED_EVENTS
    g1: getInstance(alg);
    g2: getInstance(alg, _);
    Gets := g1, g2;

  i1: init(keySize);
  i2: init(keySize, _);
  i3: init(_);
  i4: init(_, _);
  Inits := i1, i2, i3, i4;

  gk: key = generateKey();

ENFORCES_ORDER
    Gets, Inits?, gk

ENFORCES_CONSTRAINTS
 alg in {"AES"} => keySize in {128, 192, 256};
 alg in {"DES"} => keySize in {56};
 alg in {"Blowfish"} => keySize in {40, 44, 48,
52, 56, ..., 436, 440};

ENSURES
    generatedKey(key,alg);
```

Crypto APIs

- Studies show: Hard to use
- Solution:

```
KeyGenerator keyGen = KeyGenerator.getInstance("AES");
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```



```
KeyGenerator keyGen = KeyGenerator.getInstance("AES");
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

```
SPEC javax.crypto.KeyGenerator
USES_OBJECTS
    int keySize;
    javax.crypto.SecretKey key;
    java.lang.String alg;

REQUIRED_EVENTS
    g1: getInstance(alg);
    g2: getInstance(alg, _);
    Gets := g1, g2;

  i1: init(keySize);
  i2: init(keySize, _);
  i3: init(_);
  i4: init(_, _);
  Inits := i1, i2, i3, i4;

  gk: key = generateKey();

ENFORCES_ORDER
    Gets, Inits?, gk

ENFORCES_CONSTRAINTS
 alg in {"AES"} => keySize in {128, 192, 256};
 alg in {"DES"} => keySize in {56};
 alg in {"Blowfish"} => keySize in {40, 44, 48,
52, 56, ..., 436, 440};

ENSURES
    generatedKey(key,alg);
```

Crypto APIs & Variability

Methods & Overloads

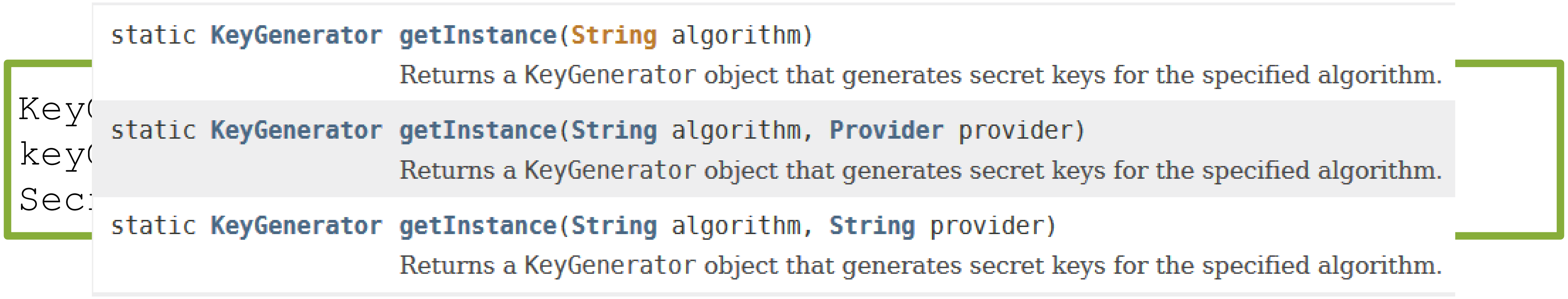
```
KeyGenerator keyGen = KeyGenerator.getInstance(“AES”);
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

Deutsche Forschungsgemeinschaft

CROSSING

Crypto APIs & Variability

Methods & Overloads



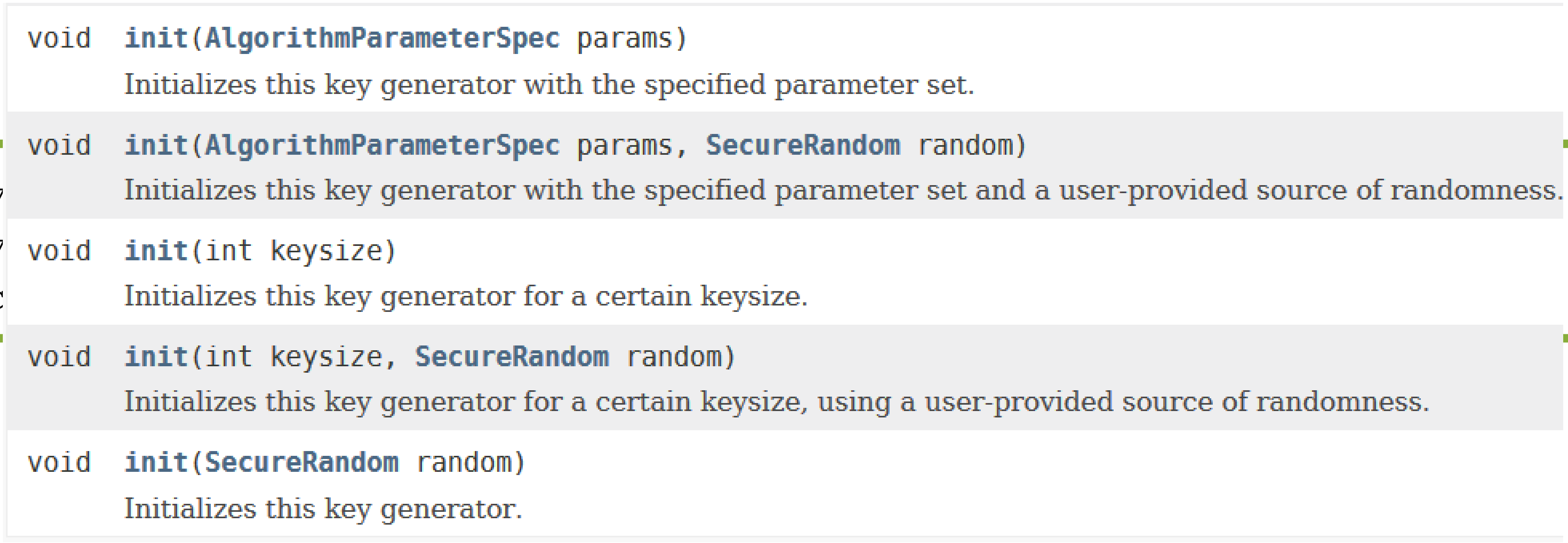
```
Key(
key(
Sec
```

```
static KeyGenerator getInstance(String algorithm)
                    Returns a KeyGenerator object that generates secret keys for the specified algorithm.
static KeyGenerator getInstance(String algorithm, Provider provider)
                    Returns a KeyGenerator object that generates secret keys for the specified algorithm.
static KeyGenerator getInstance(String algorithm, String provider)
                    Returns a KeyGenerator object that generates secret keys for the specified algorithm.
```

Crypto APIs & Variability

Methods & Overloads

```
Key
key
Sec
```



void	`init(AlgorithmParameterSpec params)` Initializes this key generator with the specified parameter set.
void	`init(AlgorithmParameterSpec params, SecureRandom random)` Initializes this key generator with the specified parameter set and a user-provided source of randomness.
void	`init(int keysize)` Initializes this key generator for a certain keysize.
void	`init(int keysize, SecureRandom random)` Initializes this key generator for a certain keysize, using a user-provided source of randomness.
void	`init(SecureRandom random)` Initializes this key generator.

DSL - Sample Specification – KeyGenerator

```
KeyGenerator keyGen = KeyGenerator.getInstance("AES");
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

DSL - Sample Specification – KeyGenerator

```
KeyGenerator keyGen = KeyGenerator.getInstance(“AES”);
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

```
SPEC javax.crypto.KeyGenerator
```

DSL - Sample Specification – KeyGenerator

```
KeyGenerator keyGen = KeyGenerator.getInstance("AES");
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

```
SPEC javax.crypto.KeyGenerator
USES_OBJECTS
    int keySize;
    javax.crypto.SecretKey key;
    java.lang.String alg;

REQUIRED_EVENTS
    g1: getInstance(alg);
    g2: getInstance(alg, _);
```

```
i1: init(keySize);
i2: init(keySize, _);
i3: init(_);
i4: init(_, _);

gk: key = generateKey();
```

Crypto APIs & Variability

Usage Patterns

```
KeyGenerator keyGen = KeyGenerator.getInstance("AES");
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

Crypto APIs & Variability

Usage Patterns

```
KeyGenerator keyGen = KeyGenerator.getInstance("AES");

SecretKey key = keyGen.generateKey();
```

DSL - Sample Specification – KeyGenerator – contd.

```
KeyGenerator keyGen = KeyGenerator.getInstance(“AES”);
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

```
SPEC javax.crypto.KeyGenerator
USES_OBJECTS
    int keySize;
    javax.crypto.SecretKey key;
    java.lang.String alg;

REQUIRED_EVENTS
    g1: getInstance(alg);
    g2: getInstance(alg, _);
    Gets := g1, g2;

    i1: init(keySize);
    i2: init(keySize, _);
    i3: init(_);
    i4: init(_, _);
    Inits := i1, i2, i3, i4;

    gk: key = generateKey();

ENFORCES_ORDER
    Gets, Inits?, gk
```

Crypto APIs & Variability

Parameter Values & Dependencies between them

AES, DES, Blowfish, …

```
KeyGenerator keyGen = KeyGenerator.getInstance(“AES”);
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

56, 128, 192, 256, 512, 1024, 2048, 4096, …

DSL - Sample Specification – KeyGenerator – contd.

```
KeyGenerator keyGen = KeyGenerator.getInstance("AES");
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

```
SPEC javax.crypto.KeyGenerator
USES_OBJECTS
    int keySize;
    javax.crypto.SecretKey key;
    java.lang.String alg;

REQUIRED_EVENTS
    g1: getInstance(alg);
    g2: getInstance(alg, _);
    Gets := g1, g2;
```

```
  i1: init(keySize);
  i2: init(keySize, _);
  i3: init(_);
  i4: init(_, _);
  Inits := i1, i2, i3, i4;

  gk: key = generateKey();

ENFORCES_ORDER
    Gets, Inits?, gk

ENFORCES_CONSTRAINTS
 alg in {"AES"} => keySize in {128, 192, 256};
 alg in {"DES"} => keySize in {56};
 alg in {"Blowfish"} => keySize in {40, 44, 48,
52, 56, ..., 436, 440};
```

Crypto APIs & Variability

Interaction between classes

```
KeyGenerator keyGen = KeyGenerator.getInstance(“AES”);
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

Crypto APIs & Variability

Interaction between classes

```
KeyGenerator keyGen = KeyGenerator.getInstance(“AES”);
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(Cipher.Encrypt_Mode, key, IV);
cipher.doFinal(data);
```

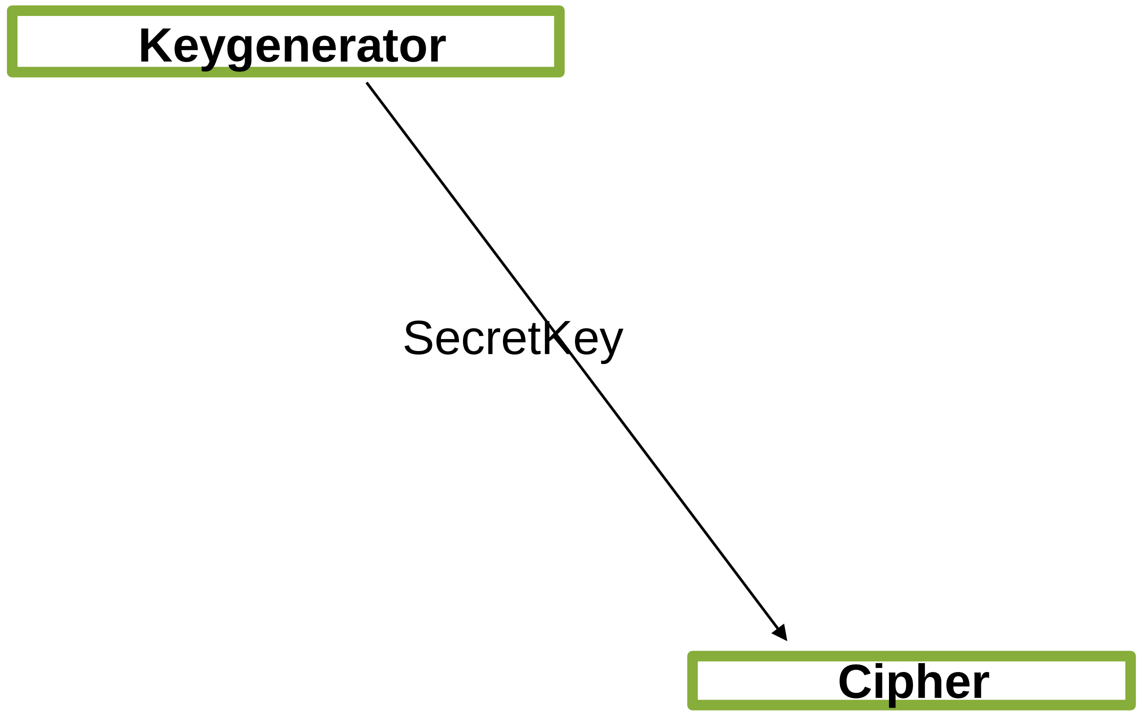
Crypto APIs & Variability
Interaction between classes

```
KeyGenerator keyGen = KeyGenerator.getInstance(“AES”);
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(Cipher.Encrypt_Mode, key, IV);
cipher.doFinal(data);
```

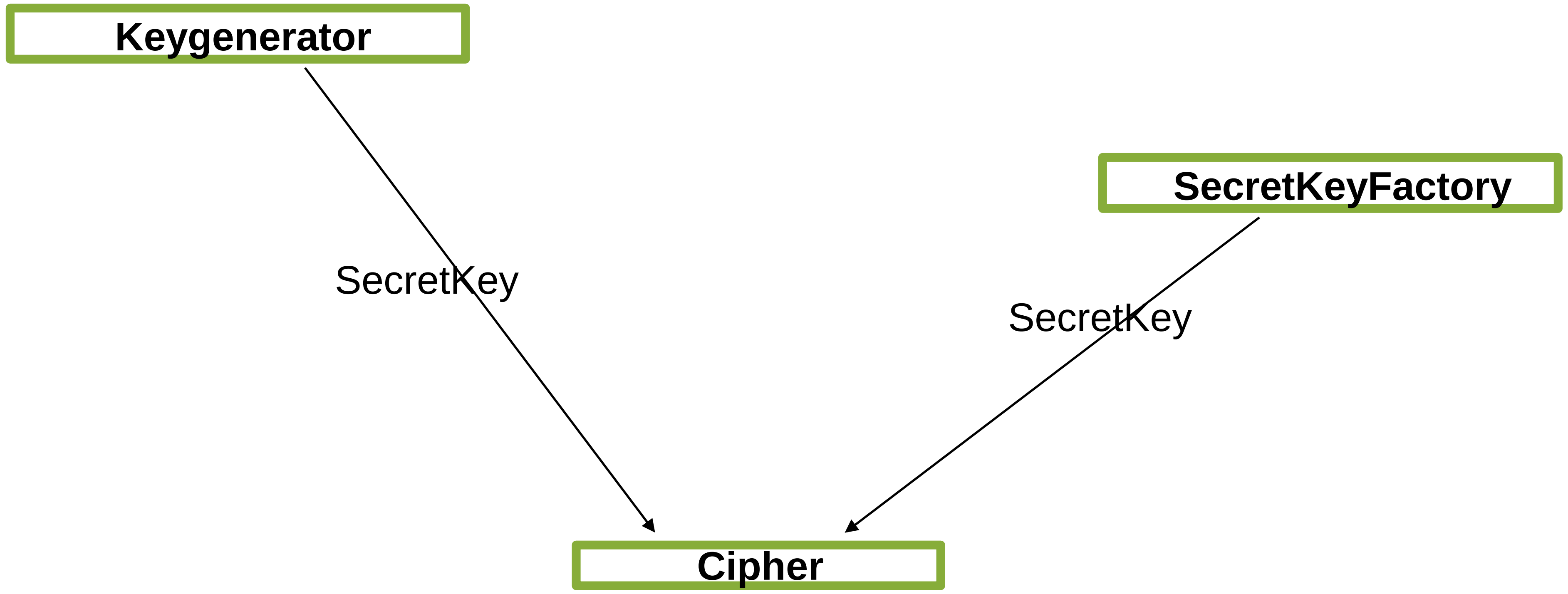
Crypto APIs & Variability

Interaction between classes



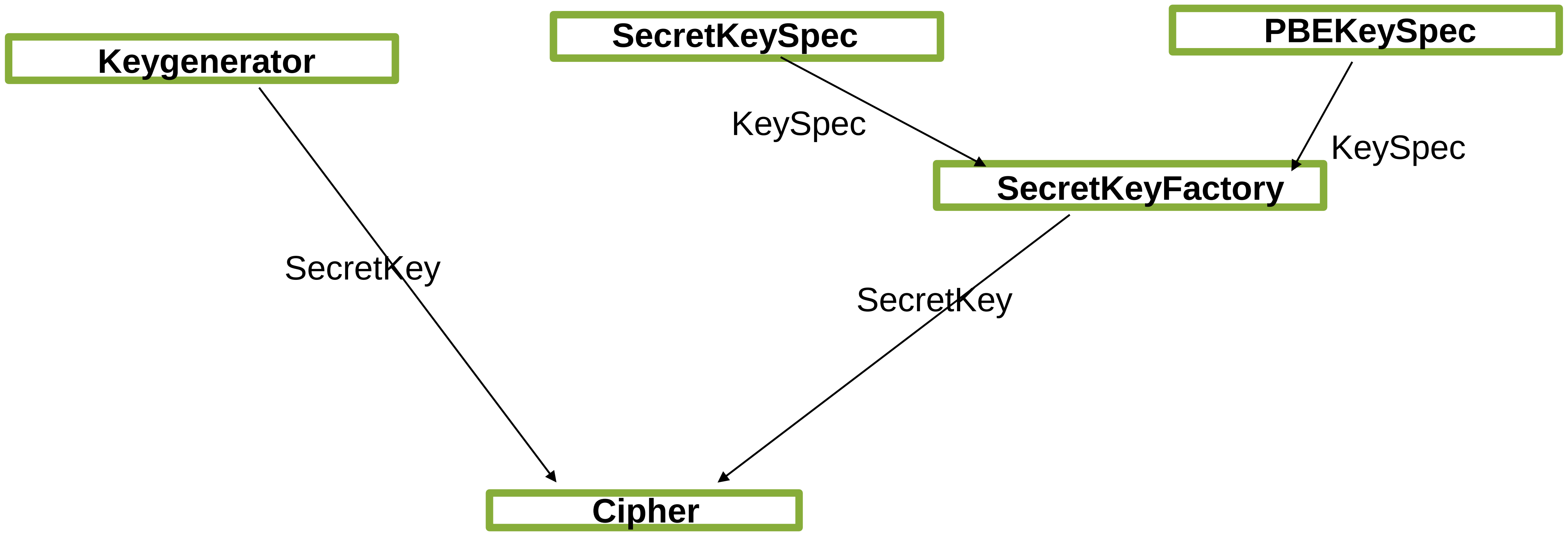
Crypto APIs & Variability

Interaction between classes



Crypto APIs & Variability

Interaction between classes



DSL - Sample Specification – KeyGenerator

```
KeyGenerator keyGen = KeyGenerator.getInstance(“AES”);
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

```
SPEC javax.crypto.KeyGenerator
USES_OBJECTS
    int keySize;
    javax.crypto.SecretKey key;
    java.lang.String alg;

REQUIRED_EVENTS
    g1: getInstance(alg);
    g2: getInstance(alg, _);
    Gets := g1, g2;
```

```
  i1: init(keySize);
  i2: init(keySize, _);
  i3: init(_);
  i4: init(_, _);
  Inits := i1, i2, i3, i4;

  gk: key = generateKey();

ENFORCES_ORDER
    Gets, Inits?, gk

ENFORCES_CONSTRAINTS
 alg in {"AES"} => keySize in {128, 192, 256};
 alg in {"DES"} => keySize in {56};
 alg in {"Blowfish"} => keySize in {40, 44, 48,
52, 56, ..., 436, 440};

ENSURES
    generatedKey(key,alg);
```

Deutsche Forschungsgemeinschaft

CROSSING

DSL - Sample Specification - Cipher

```
Cipher ciph = Cipher.getInstance("AES/CBC/PKCS5Padding");
ciph.init(Cipher.Encrypt_Mode, key, IV);
ciph.doFinal(data);
```

```
SPEC javax.crypto.Cipher
USES_OBJECTS
    java.lang.String transformation;
    int encmode;
    javax.crypto.SecretKey key;
    java.security.spec.
          AlgorithmParameterSpec params;
    java.security.
          AlgorithmParameters param;

    int pre_plain_off;
    int pre_ciphertext_off;
    int plain_off;
    int ciphertext_off;

    byte[] pre_plaintext;
    byte[] pre_ciphertext;

    byte[] plainText;
    byte[] cipherText;
    java.nio.ByteBuffer plainBuffer;
    java.nio.ByteBuffer cipherBuffer;
```

```
REQUIRED_EVENTS
    g1: getInstance(transformation);
    g2:getInstance(transformation, _);
    Gets := g1, g2;

    i1: init(encmode, key);
    i2: init(encmode, _);
    i3: init(encmode, key, _);
    i4: init(encmode, key, param);
    i5: init(encmode, key, params, _);
    i6: init(encmode, key, param, _);
    i7: init(encmode, key, _);
    i8: init(encmode, _, _);
    IWOIV := i2, i8, i1, i7;
    IWIV := i3, i4, i5, i6;
    Inits := IWOIV, IWIV;
```

DSL - Sample Specification - Cipher

```
Cipher ciph = Cipher.getInstance("AES/CBC/PKCS5Padding");
ciph.init(Cipher.Encrypt_Mode, key, IV);
ciph.doFinal(data);
```

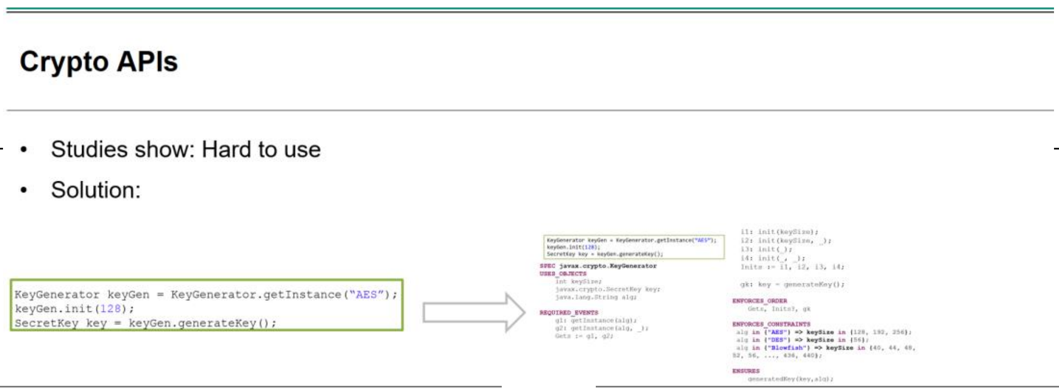
```
u1: pre_ciphertext = update(pre_plaintext);
u2: pre_ciphertext = update(pre_plaintext, pre_plain_off, _);
u3: update(pre_plaintext, pre_plain_off, pre_len, pre_ciphertext);
u4: update(pre_plaintext, pre_plain_off, pre_len, pre_ciphertext, pre_ciphertext_off);
u5: update(pre_plaintext, pre_ciphertext);
updates := u1, u2, u3, u4, u5;

f1: cipherText = doFinal();
f2: cipherText =  doFinal(plainText);
f3: doFinal(cipherText, plain_off);
f4: cipherText = doFinal(cipherText, plain_off, len);
f5: doFinal(plainText, plain_off, len, cipherText);
f6: doFinal(plainText, plain_off, len, cipherText, ciphertext_off);
f7: doFinal(plainBuffer, cipherBuffer);
FINWOU := f2, f5, f6, f7;
DOFINALS := FINWOU, f1, f3, f4;

ENFORCES_ORDER
    Gets, Inits, (FINWOU | (updates+, DOFINALS))+
```

DSL - Sample Specification - Cipher

```
Cipher ciph = Cipher.getInstance("AES/CBC/PKCS5Padding");
ciph.init(Cipher.Encrypt_Mode, key, IV);
ciph.doFinal(data);
```

```
ENFORCES_CONSTRAINTS
  alg(transformation) in {"AES", "Blowfish"}
  alg(transformation) in {"AES"} => mode(transformation) in {"CBC", "PCBC", "CTR", "CTS", "CFB"}
  alg(transformation) in {"Blowfish", "DESede"} => mode(transformation) in {"CBC", "PCBC", "CTR"}
  alg(transformation) in {"RSA"} => mode(transformation) in {"ECB"} && pad(transformation) in {"OAEPWithMD5AndMGF1Padding"}

  alg(transformation) in {"AES", "Blowfish", "DESede", "RC2"} => pad(transformation) in {"NoPadding", "PKCS5Padding"}

  mode(transformation) in {"CBC"} && encmode != 1 => noCallOf(IWOIV);

  generatedKey(key, alg(transformation));

  encmode >= 1 && encmode <= 4;
  pre_plaintext.length >= pre_plain_off + len;
  pre_ciphertext.length <= pre_ciphertext_off;
  plainText.length <= plain_off + len;
  cipherText <= ciphertext_off;

ENSURES
    encrypted(plainText, cipherText);
```

Summary

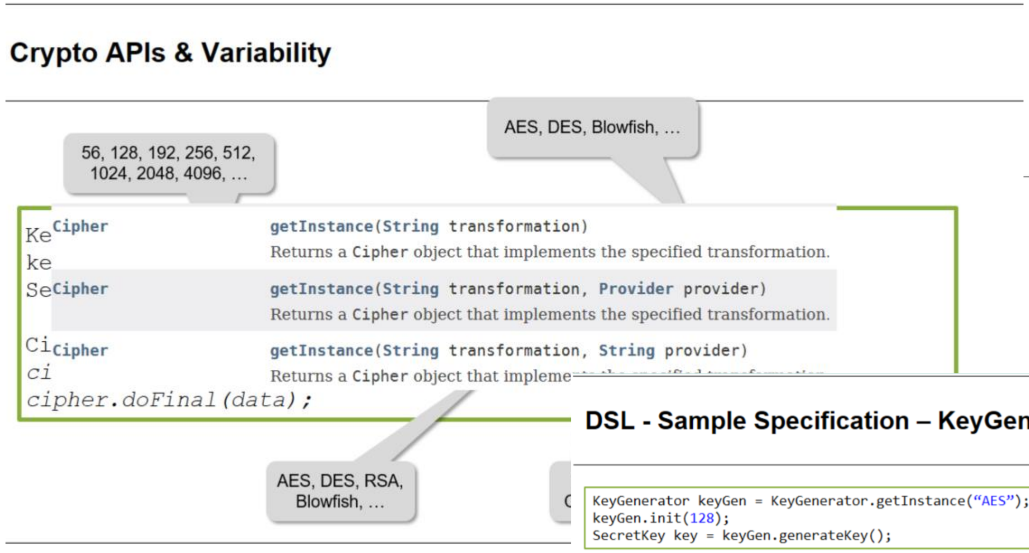


Crypto APIs

- Studies show: Hard to use
- Solution:

```
KeyGenerator keyGen = KeyGenerator.getInstance("AES");
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

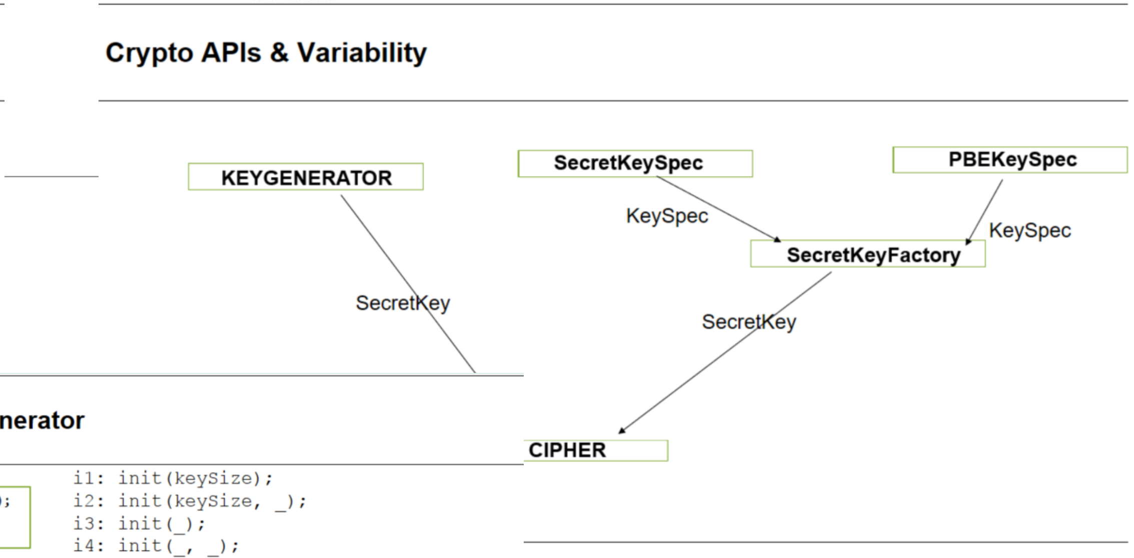
```
KeyGenerator keyGen = KeyGenerator.getInstance("AES");
keyGen.init(128);
SecretKey key = keyGen.generateKey();
SPEC javax.crypto.KeyGenerator
USES_OBJECTS
    int keySize;
    javax.crypto.SecretKey key;
    java.lang.String alg;

REQUIRED_EVENTS
    g1: getInstance(alg);
    g2: getInstance(alg, _);
    Gets := g1, g2;

    i1: init(keySize);
    i2: init(keySize, _);
    i3: init(_);
    i4: init(_, _);
    Inits := i1, i2, i3, i4;

    gk: key = generateKey();

ENFORCES_ORDER
    Gets, Inits?, gk

ENFORCES_CONSTRAINTS
 alg in {"AES"} => keySize in {128, 192, 256};
 alg in {"DES"} => keySize in {56};
 alg in {"Blowfish"} => keySize in {40, 44, 48,
52, 56, ..., 436, 440};

ENSURES
    generatedKey(key,alg);
```



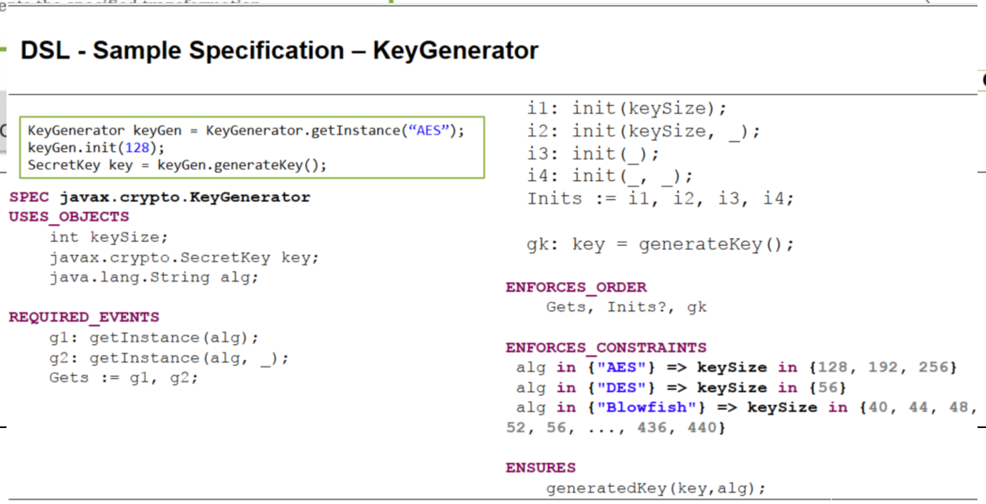
Crypto APIs & Variability

```
Ke
ke
Se
Ci
ci
cipher.doFinal(data);
```

Cipher	getInstance(String transformation) Returns a Cipher object that implements the specified transformation.
Cipher	getInstance(String transformation, Provider provider) Returns a Cipher object that implements the specified transformation.
Cipher	getInstance(String transformation, String provider) Returns a Cipher object that impleme



Crypto APIs & Variability



DSL - Sample Specification – KeyGenerator

```
KeyGenerator keyGen = KeyGenerator.getInstance("AES");
keyGen.init(128);
SecretKey key = keyGen.generateKey();
```

```
SPEC javax.crypto.KeyGenerator
USES_OBJECTS
    int keySize;
    javax.crypto.SecretKey key;
    java.lang.String alg;

REQUIRED_EVENTS
    g1: getInstance(alg);
    g2: getInstance(alg, _);
    Gets := g1, g2;

    i1: init(keySize);
    i2: init(keySize, _);
    i3: init(_);
    i4: init(_, _);
    Inits := i1, i2, i3, i4;

    gk: key = generateKey();

ENFORCES_ORDER
    Gets, Inits?, gk

ENFORCES_CONSTRAINTS
 alg in {"AES"} => keySize in {128, 192, 256}
 alg in {"DES"} => keySize in {56}
 alg in {"Blowfish"} => keySize in {40, 44, 48,
52, 56, ..., 436, 440}

ENSURES
    generatedKey(key,alg);
```