RubyKaigi2006 セキュアアプリケーションプログラミング

2006-06-10

なひ（サリオンシステムズリサーチ）

SARION® SYSTEMS RESEARCH

目次

- セキュリティの基礎技術
  - 暗号化
  - 認証 + 完全性の保証（改ざんがないことの確認）
- アプリケーションへの応用
  - サンプルアプリケーションを使い、単純な通信から、認証、暗号化の追加拡張（ステップ・バイ・ステップで）
  - OpenPGPツールキットでできること

以上を、Rubyのサンプルソースコードを元に紹介

`http://dev.ctor.org/download/rubykaigi2006.tar.gz`

SARION® SYSTEMS RESEARCH

目的

- セキュリティ基礎技術を使う際のサンプルコードを提供。落とし穴にハマらないように。
- アプリケーションへ応用できるコードを提供する。
- 誰かOpenPGPツールキットの開発を一緒にやりませんか。

SARION® SYSTEMS RESEARCH

セキュリティの基礎機能

- 各種セキュリティサービスの基礎となっているのは、以下の3つの機能
  - 暗号化
  - 認証
  - 完全性の保証（改ざんがないことの確認）

※通常、認証と完全性の保証はペア

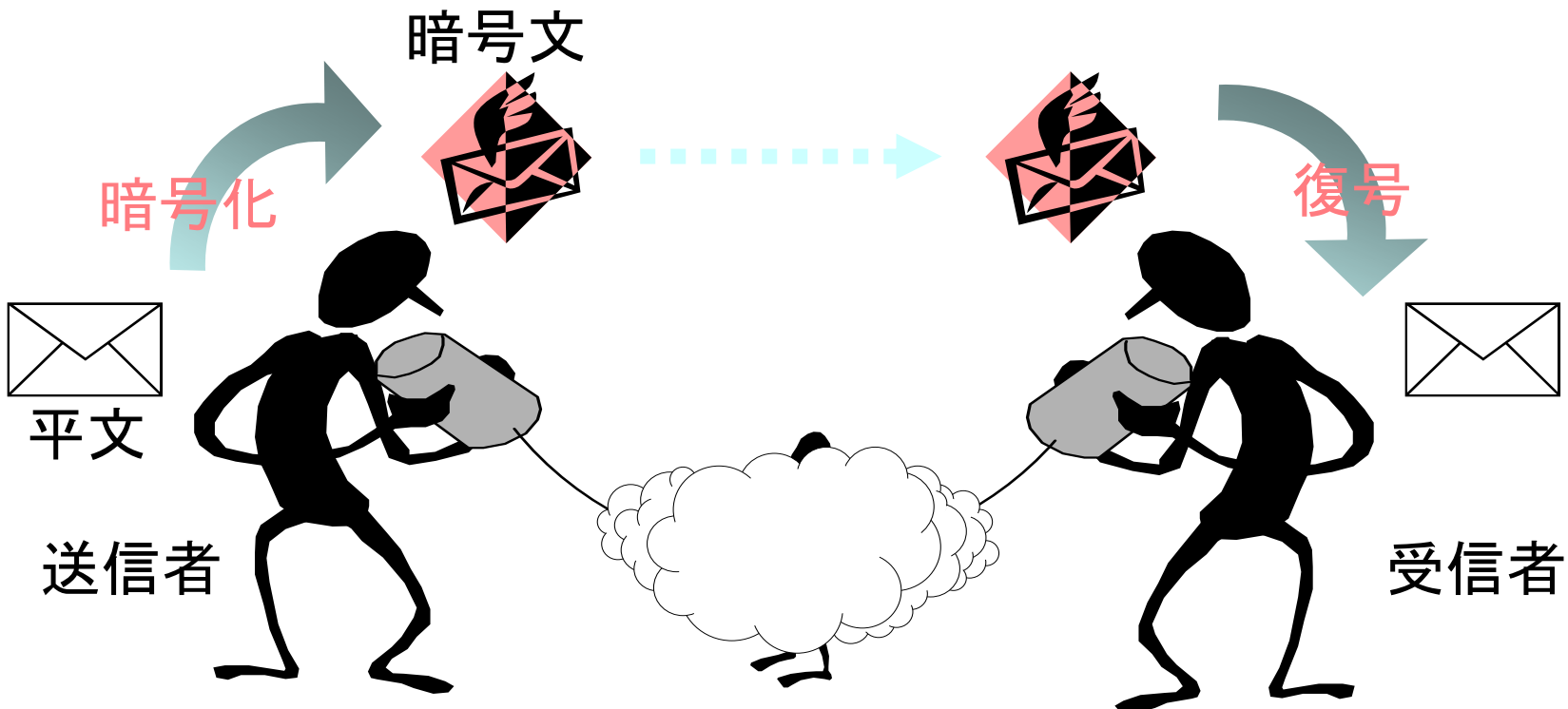
SARION® SYSTEMS RESEARCH

セキュリティの基礎技術

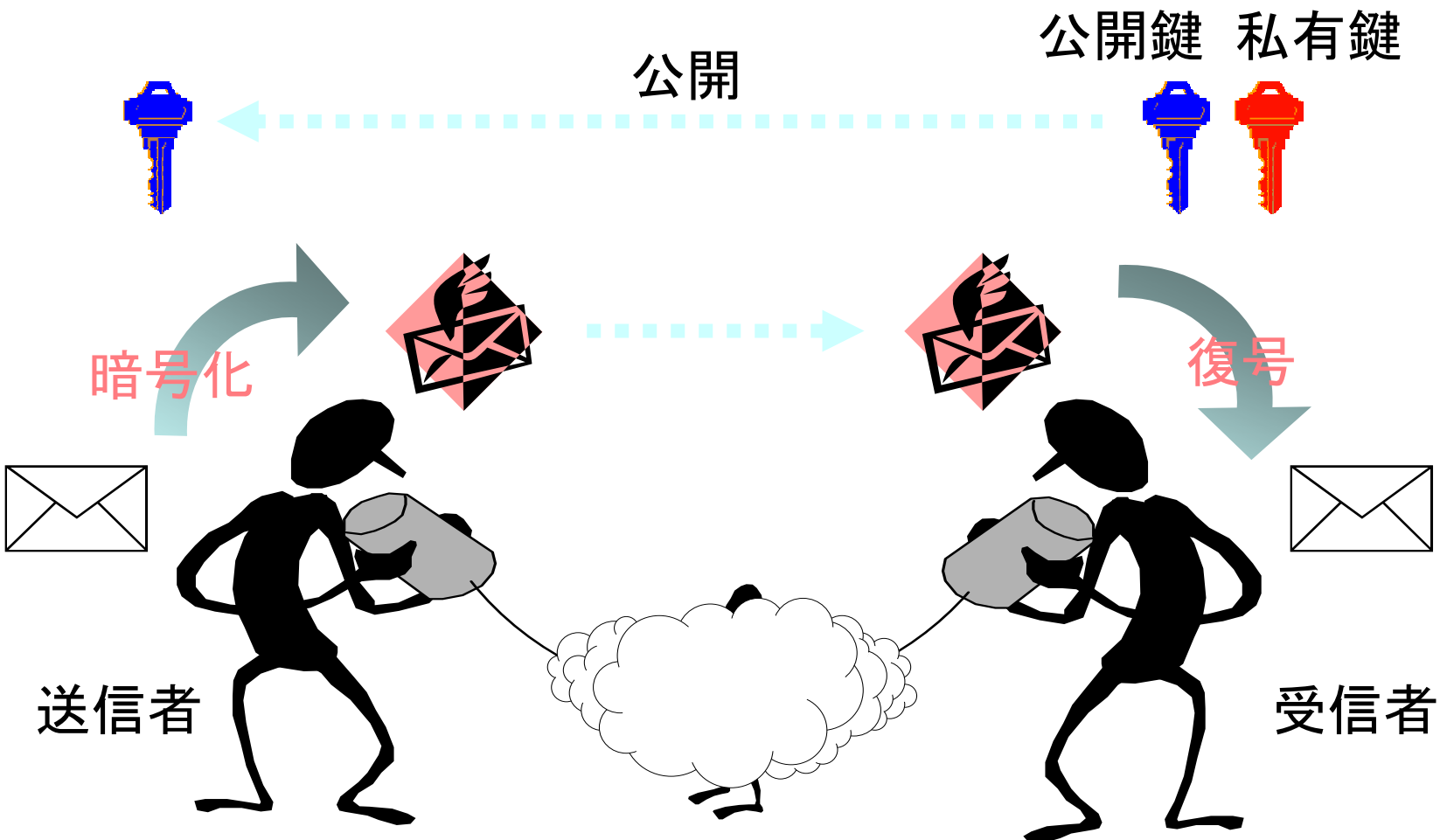
- 3つのセキュリティ基礎機能を実現するための、3つの基礎技術
  - 秘密鍵暗号（対称鍵暗号、共通鍵暗号）
  - 公開鍵暗号（非対称鍵暗号）
  - ハッシュ関数：一方向な値の変換（誰でも計算できる・誰も元には戻せない）

SARION® SYSTEMS RESEARCH

対称鍵暗号
（共通鍵暗号、秘密鍵暗号）



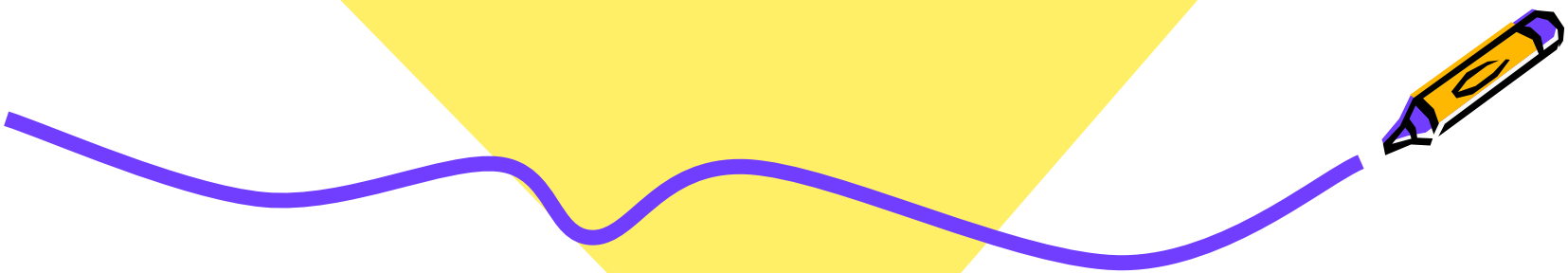
非対称鍵暗号(公開鍵暗号)



暗号鍵の種類

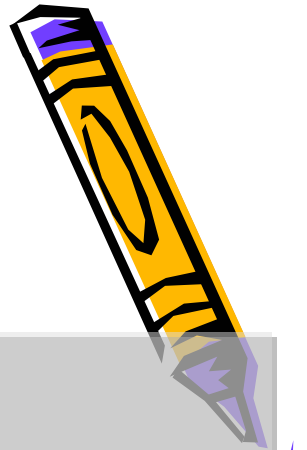
- 暗号鍵には3種類ある
  - 秘密鍵（secret key） seckey
    - 対称鍵暗号で利用。共通鍵とも呼ばれる。秘密。通常、他人と共有しなければならない。
  - 私有鍵（private key） privkey
    - 非対称鍵暗号で利用。秘密。他人に知られてはならない（private）。公開鍵とペア。
  - 公開鍵（public key） pubkey
    - 非対称鍵暗号で利用。公開。他人に知られなくてはならない。私有鍵とペア。

11_暗号化



SARION®
SYSTEMS
RESEARCH

111_素朴なAES(暗号化)



```
# 秘密鍵を読み込む
key = File.read("seckey.bin")
# ECBでAESエンジンを生成
cipher =
  OpenSSL::Cipher::Cipher.new("AES-128-ECB")
# 初期化
cipher.encrypt
cipher.key = key
# 暗号化
print cipher.update(ARGF.read) + cipher.final
```

SARION® SYSTEMS RESEARCH

111_素朴なAES(復号)

```
# 秘密鍵を読み込む
key = File.read("seckey.bin")
# ECBでAESエンジンを生成
cipher =
  OpenSSL::Cipher::Cipher.new("AES-128-ECB")
# 初期化
cipher.decrypt
cipher.key = key
# 復号
print cipher.update(ARGF.read) + cipher.final
```

※ECBは入れ替え攻撃が可能なので注意

SARION® SYSTEMS RESEARCH

112_「らしい」AES

```
# CBCでAESエンジンを生成
cipher = OpenSSL::Cipher::Cipher.new("AES-256-
  CBC")
# IV(initial vector)を作る。AESは128bitブロック暗号
iv = OpenSSL::Random.random_bytes(16)
# 初期化
cipher.encrypt
cipher.key = key
cipher.iv = iv
# IVを渡す。知られても問題ない
print iv
# 暗号化
print cipher.update(ARGF.read) + cipher.final
```

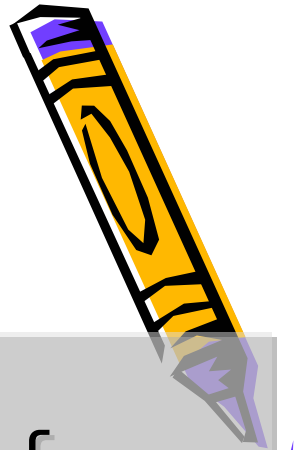
※CWC、OFBなど各種あるが、とりあえずOpenSSLではCBCを使いましょう。

SARION® SYSTEMS RESEARCH

113_パスワードによる簡単AES

```
# パスワードはどこかから持ってくる
password = File.read("password.txt")
# AESエンジンを生成
cipher = OpenSSL::Cipher::Cipher.new("AES-256-
  CBC")
# 初期化
cipher.encrypt
# パスワードからseckeyとIVを生成させる ← 簡単
cipher.pkcs5_keyivgen(password)
# 暗号化
print cipher.update(ARGF.read) + cipher.final
```

115_素朴なRSA暗号(鍵生成)



```
# 鍵ペア(公開鍵、私有鍵)を生成
key = OpenSSL::PKey::RSA.new(2048) {
  print "." }
# 私有鍵を保存(ペアの両方を入れる)
File.open("privkey.pem", "w") do |file|
  file << key.to_pem
end
# 公開鍵を保存
File.open("pubkey.pem", "w") do |file|
  file << key.public_key.to_pem
end
```

115_素朴なRSA暗号（暗号化と復号）

```
# 公開鍵を読み込む
pubkey = OpenSSL::PKey::RSA.new(
  File.read("pubkey.pem"))
# 暗号化: 入力は鍵サイズより大きくなれない→実用×
print pubkey.public_encrypt(ARGF.read)
# 私有鍵を読み込む
privkey = OpenSSL::PKey::RSA.new(
  File.read("privkey.pem"))
# 復号
print privkey.private_decrypt(ARGF.read)
```

116_「らしい」RSA暗号

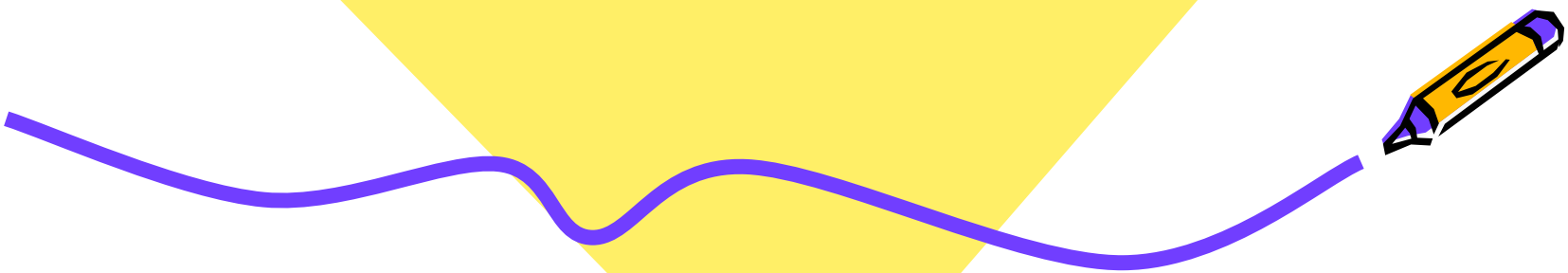
```
# AES用パスワードを動的に生成し、AESで暗号化
password = OpenSSL::Random.random_bytes(16)
cipher = OpenSSL::Cipher::Cipher.new("AES-256-CBC")
cipher.encrypt; cipher.pkcs5_keyivgen(password)
ciphertext =cipher.update(ARGF.read) + cipher.final
# パスワードをRSAで暗号化
pubkey =
  OpenSSL::PKey::RSA.new(File.read("pubkey.pem"))
cipherpassword = pubkey.public_encrypt(password)
# 暗号化したパスワードと、パスワードで暗号化した暗号文を出力
print Marshal.dump(cipherpassword) +
  Marshal.dump(ciphertext)
```

117_RSA私有鍵をパスワードで保護する

```
key = OpenSSL::PKey::RSA.new(2048) {
  print "." }
# 鍵をパスワードで保護する
# exportメソッドに秘密鍵暗号を指定する
File.open("privkey.pem", "w") do |file|
  protectedkey = key.export(
    OpenSSL::Cipher::Cipher.new(
      "AES-256-CBC"))
  file << protectedkey
end
# パスワードはコンソール上で2度入力を求められる。
```

SARION® SYSTEMS RESEARCH

12_認証+完全性の保証



SARION® SYSTEMS RESEARCH

121_ハッシュ関数による認証:HMAC

```
# 署名対称メッセージの読み込み
plain = ARGF.read
# HMACでは、秘密の共有鍵を利用する
key = File.read("seckey.bin")
# メッセージ認証コードを計算 == 署名
digester = OpenSSL::Digest::SHA1.new
sig = OpenSSL::HMAC.digest(digester,
  key, plain)
# 秘密の共有鍵を持つ検証者は、同じ計算を行い、
# 同じ署名が生成できたら、認証成功とする。
```

※HTTP-Cookiesの改ざん検知に使える

SARION® SYSTEMS RESEARCH

122_RSAによる認証(署名)

```
# 署名対象メッセージの読み込み
plain = ARGF.read
# 私有鍵を読み込む
key = OpenSSL::PKey::RSA.new(
  File.read("privkey.pem"))
# 署名
digester = OpenSSL::Digest::SHA1.new
sig = key.sign(digester, plain)
```

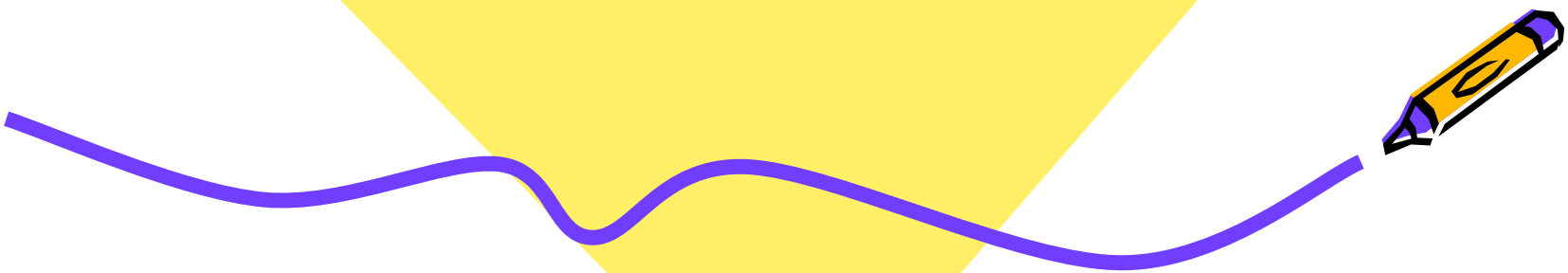
122_RSAによる認証(検証)

```
# 公開鍵の読み込み
pubkey = OpenSSL::PKey::RSA.new(
  File.read("pubkey.pem"))
# 検証対象メッセージと署名の読み込み
plain = File.read("plain.txt")
sig = File.read("plain.sig.bin")
# 検証
digester = OpenSSL::Digest::SHA1.new
if pubkey.verify(digester, sig, plain)
  puts 'authentication succeeded'
end
```

123_DSAによる認証(署名)

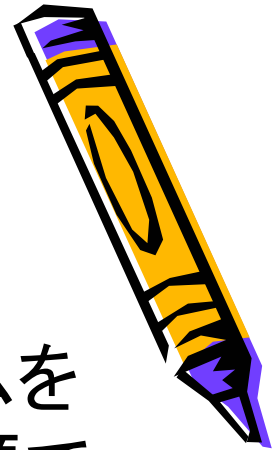
```
# 署名対象メッセージの読み込み
plain = ARGF.read
# 私有鍵を読み込む
privkey = OpenSSL::PKey::DSA.new
  (File.read("privkey.pem"))
# 署名
digester = OpenSSL::Digest::DSS1.new
sig = key.sign(digester, plain)
```

2_応用



SARION® SYSTEMS RESEARCH

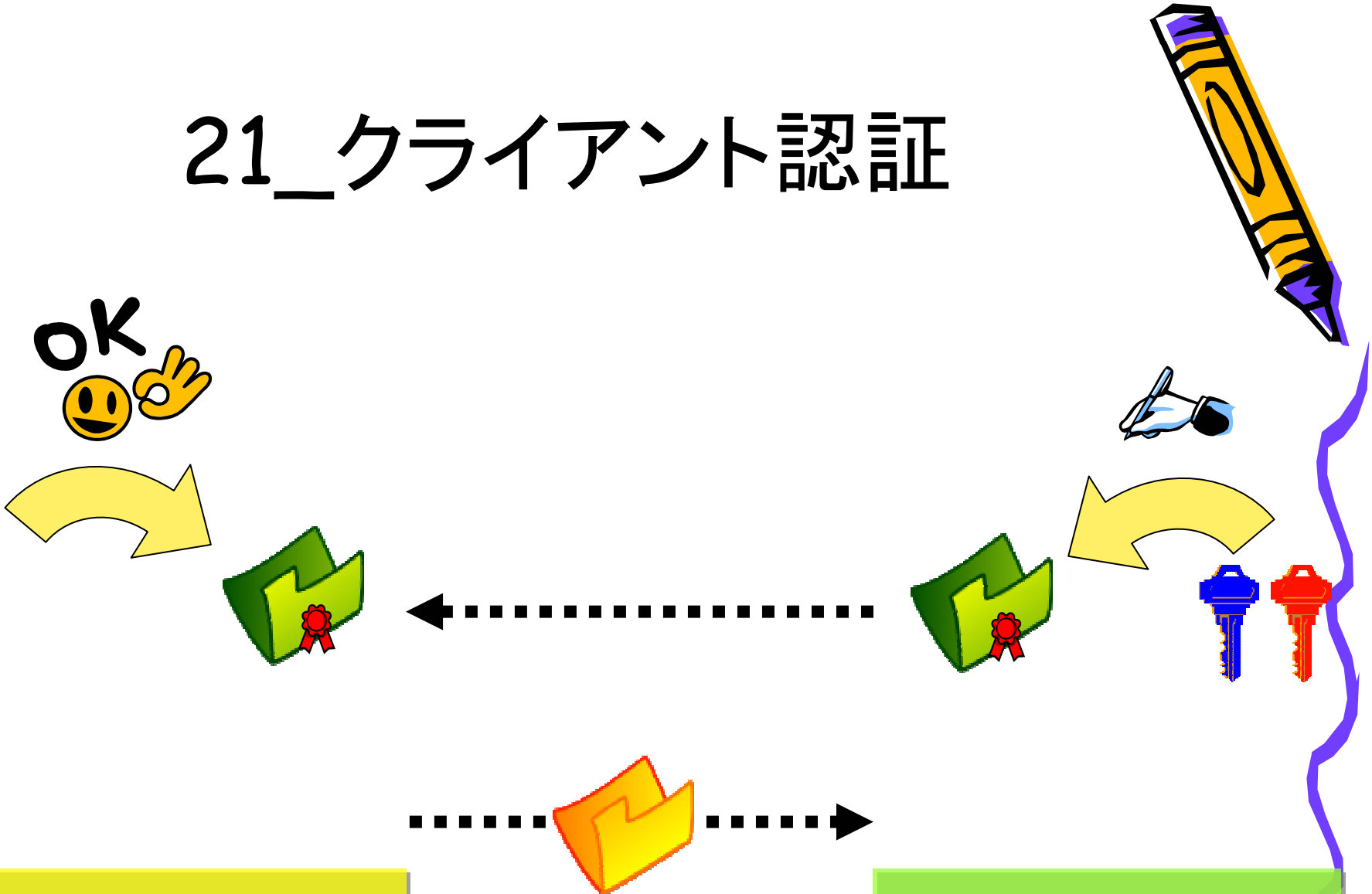
例題



- host ... 外部から与えられたRubyプログラムを評価(単にeval)し、結果を返す。今回の例題では、外部とはsocketで通信する。
- proxy ... Rubyプログラムをリモートホストに転送し、結果を取得するクライアント。

SARION®
SYSTEMS
RESEARCH

21_クライアント認証

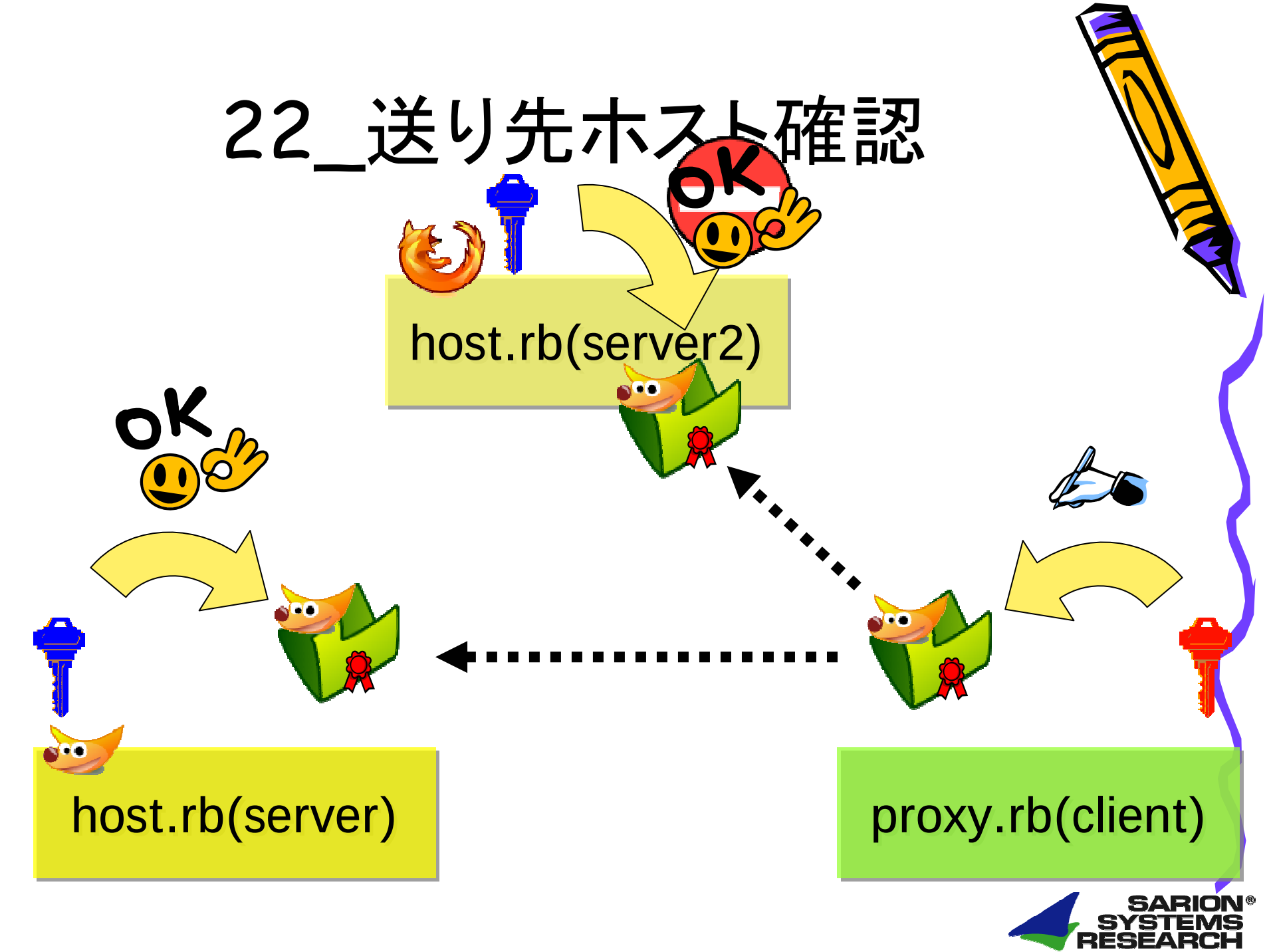


host.rb(server)

proxy.rb(client)

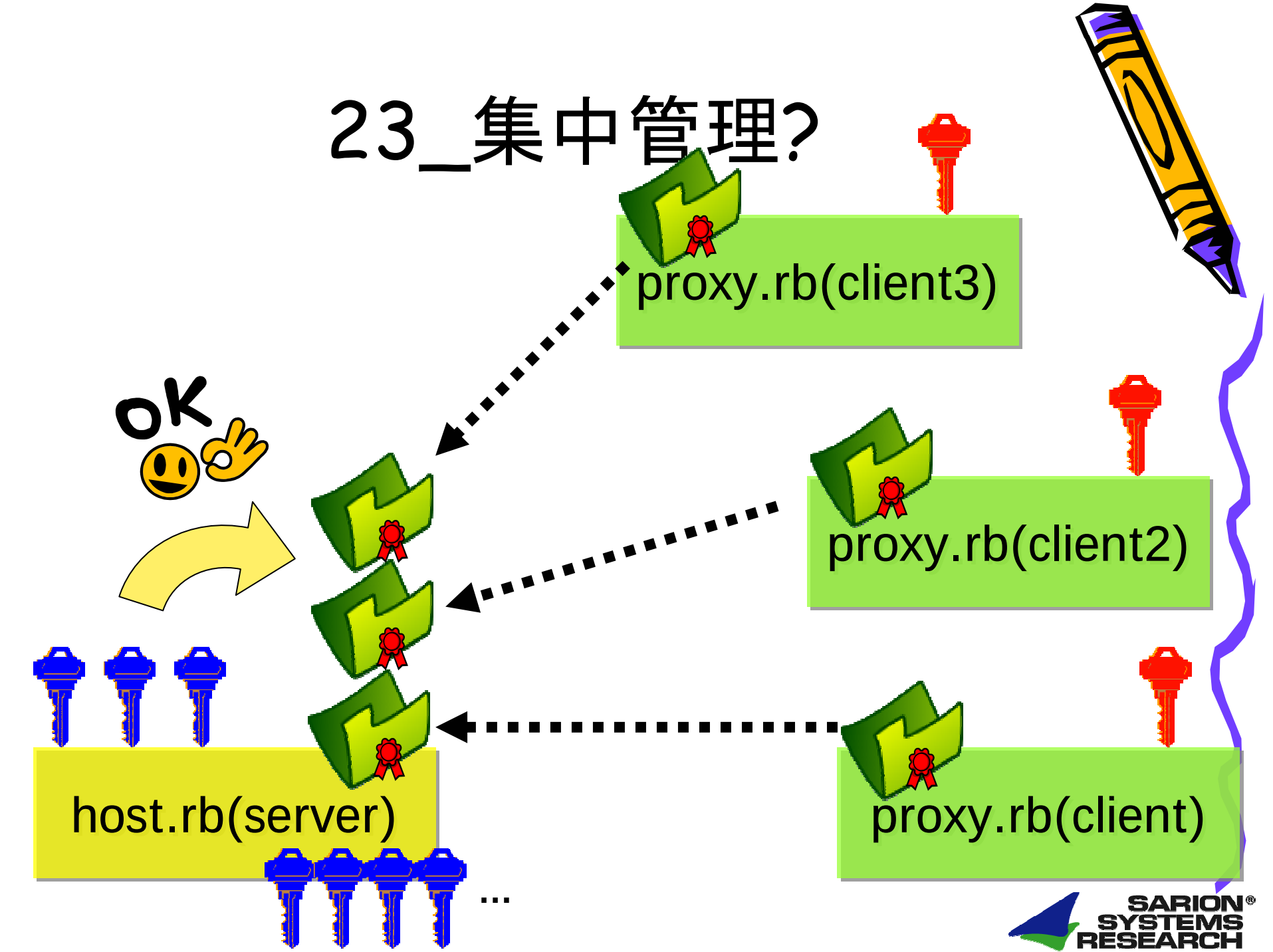
SARION® SYSTEMS RESEARCH

22_送り先ホスト確認



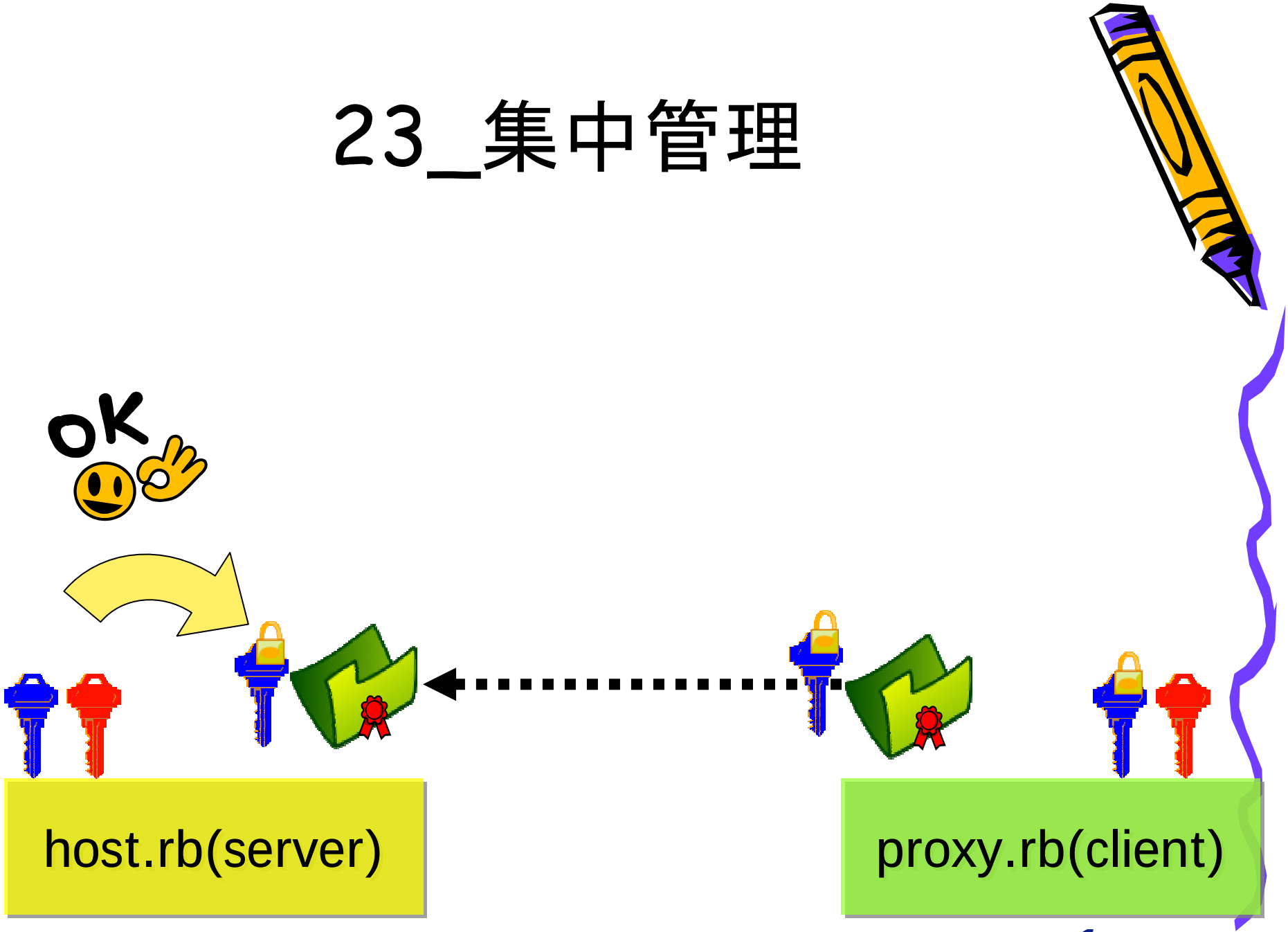
SARION® SYSTEMS RESEARCH

23_集中管理?



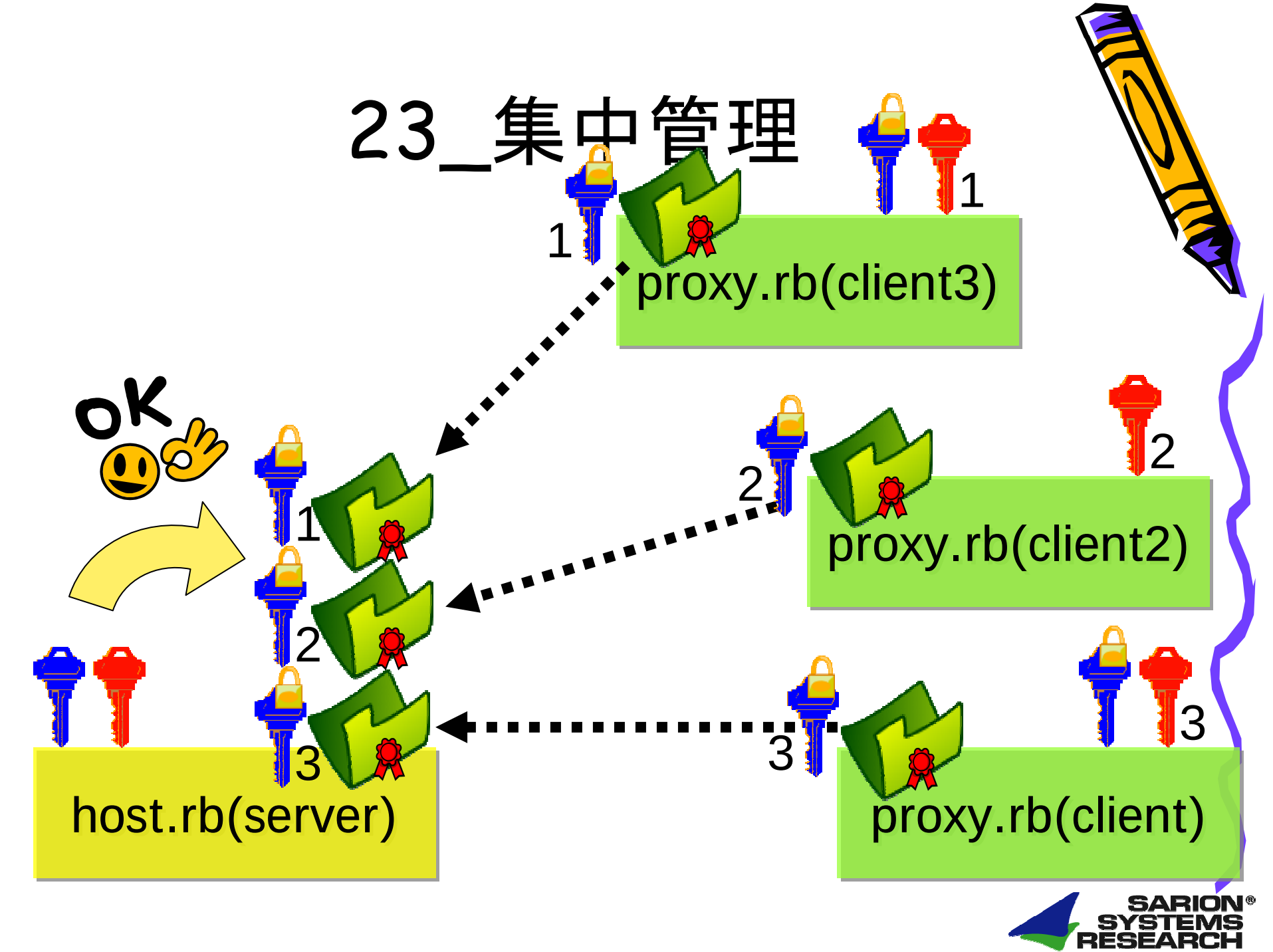
SARION® SYSTEMS RESEARCH

23_集中管理



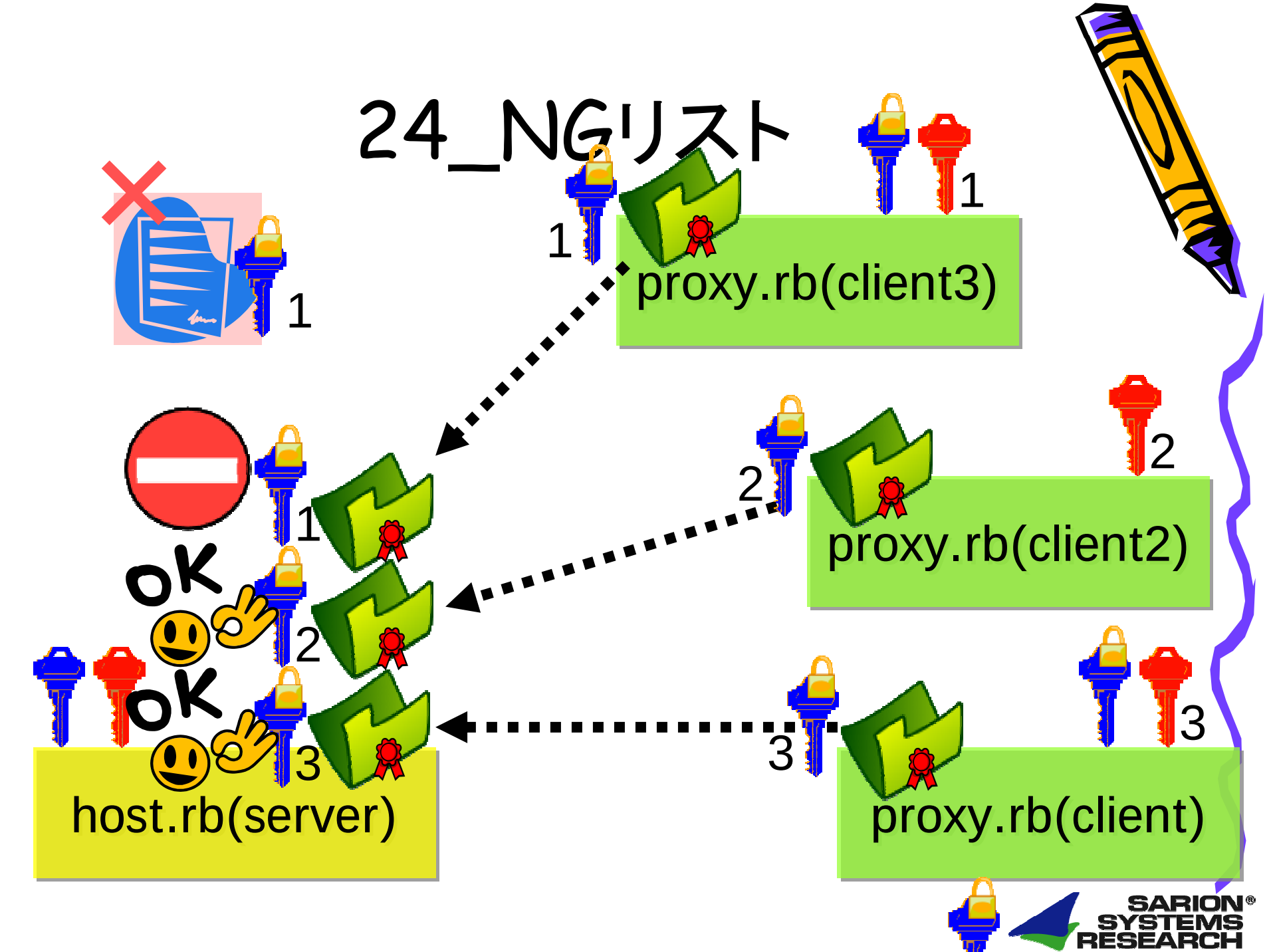
SARION® SYSTEMS RESEARCH

23_集中管理

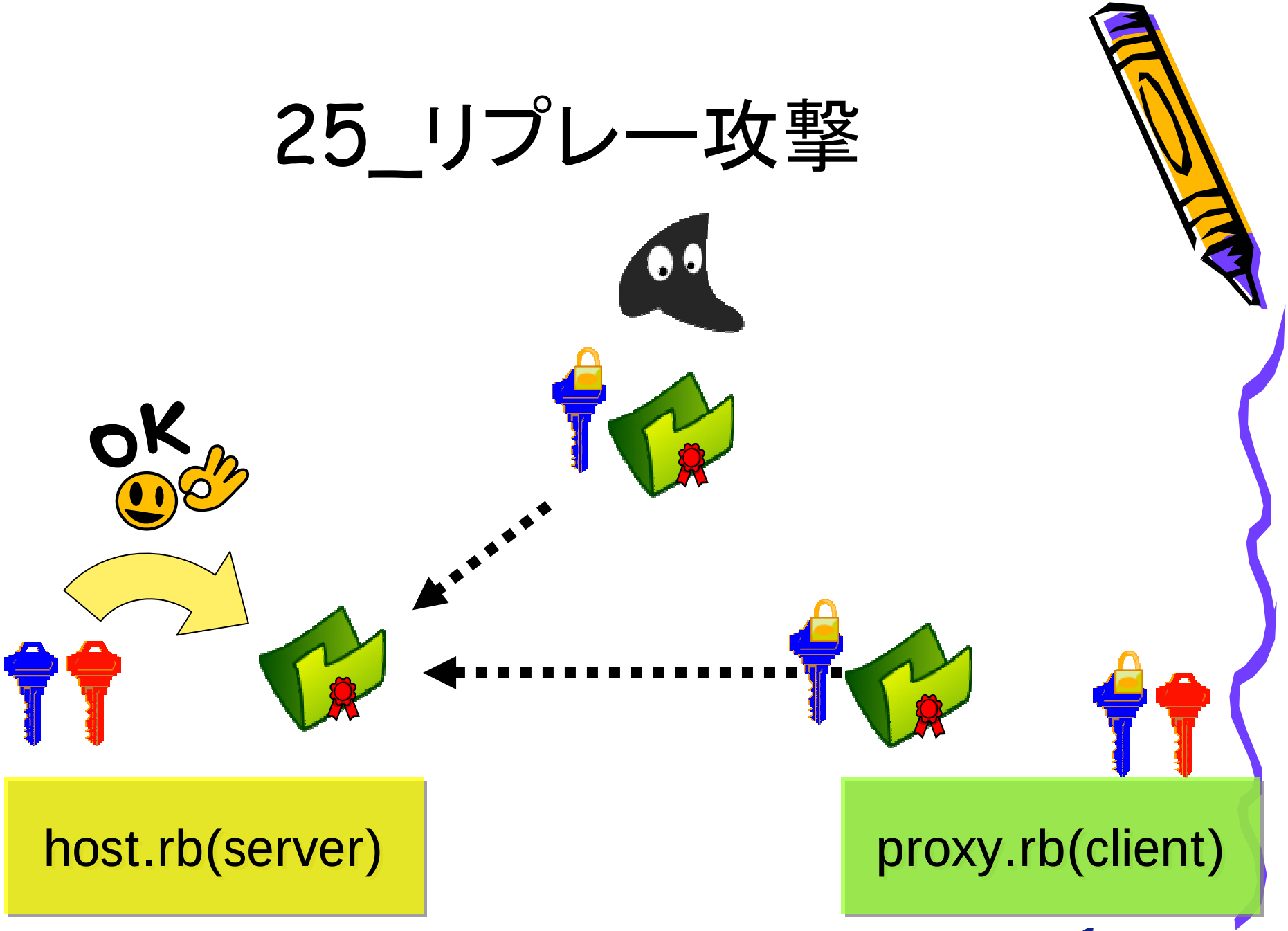


SARION® SYSTEMS RESEARCH

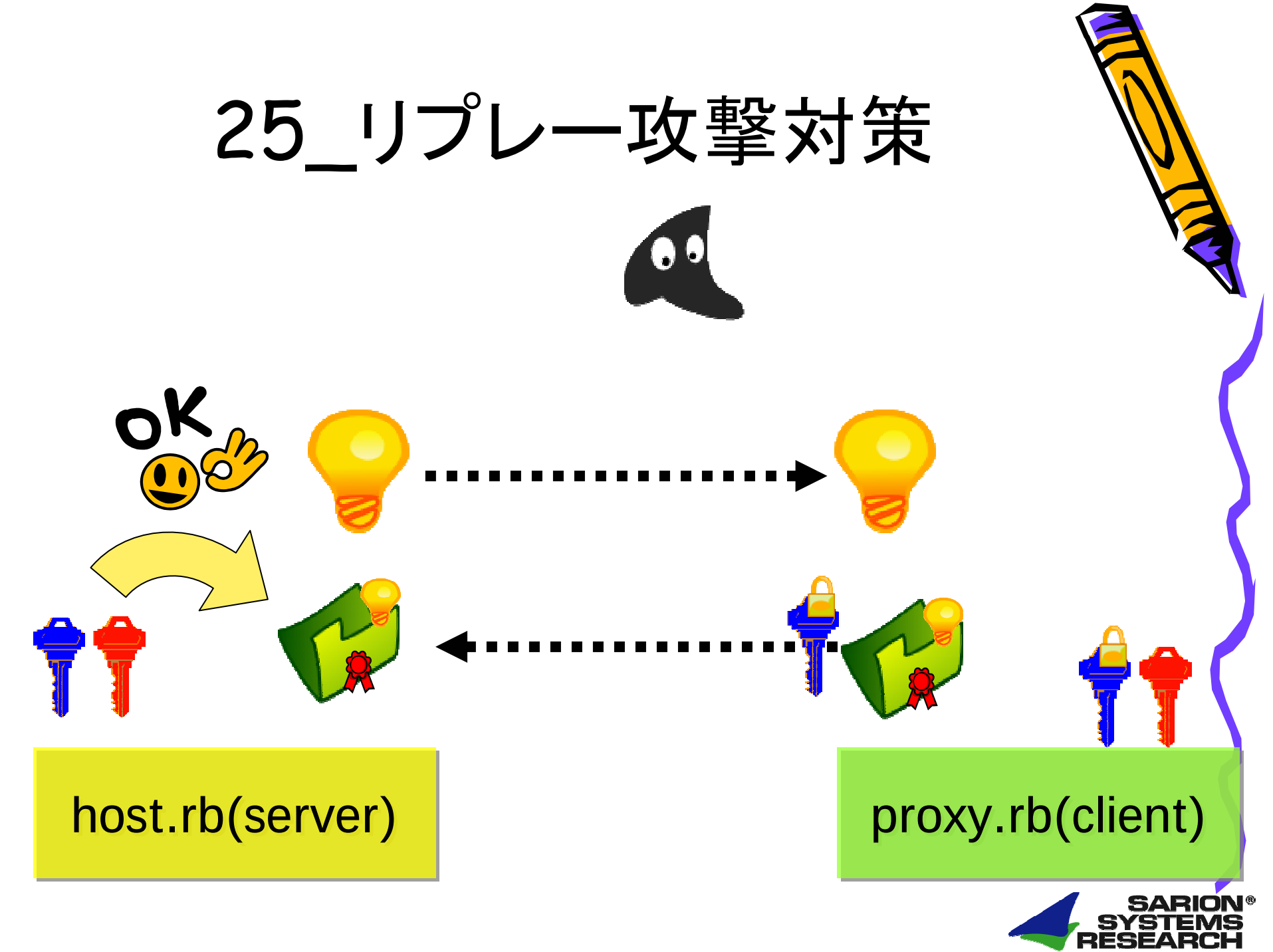
24_NGリスト


1
1
proxy.rb(client3)
1
2
2
proxy.rb(client2)
2
1
OK
2
OK
3
host.rb(server)
3
proxy.rb(client)
3
SARION®
SYSTEMS
RESEARCH

25_リプレー攻撃

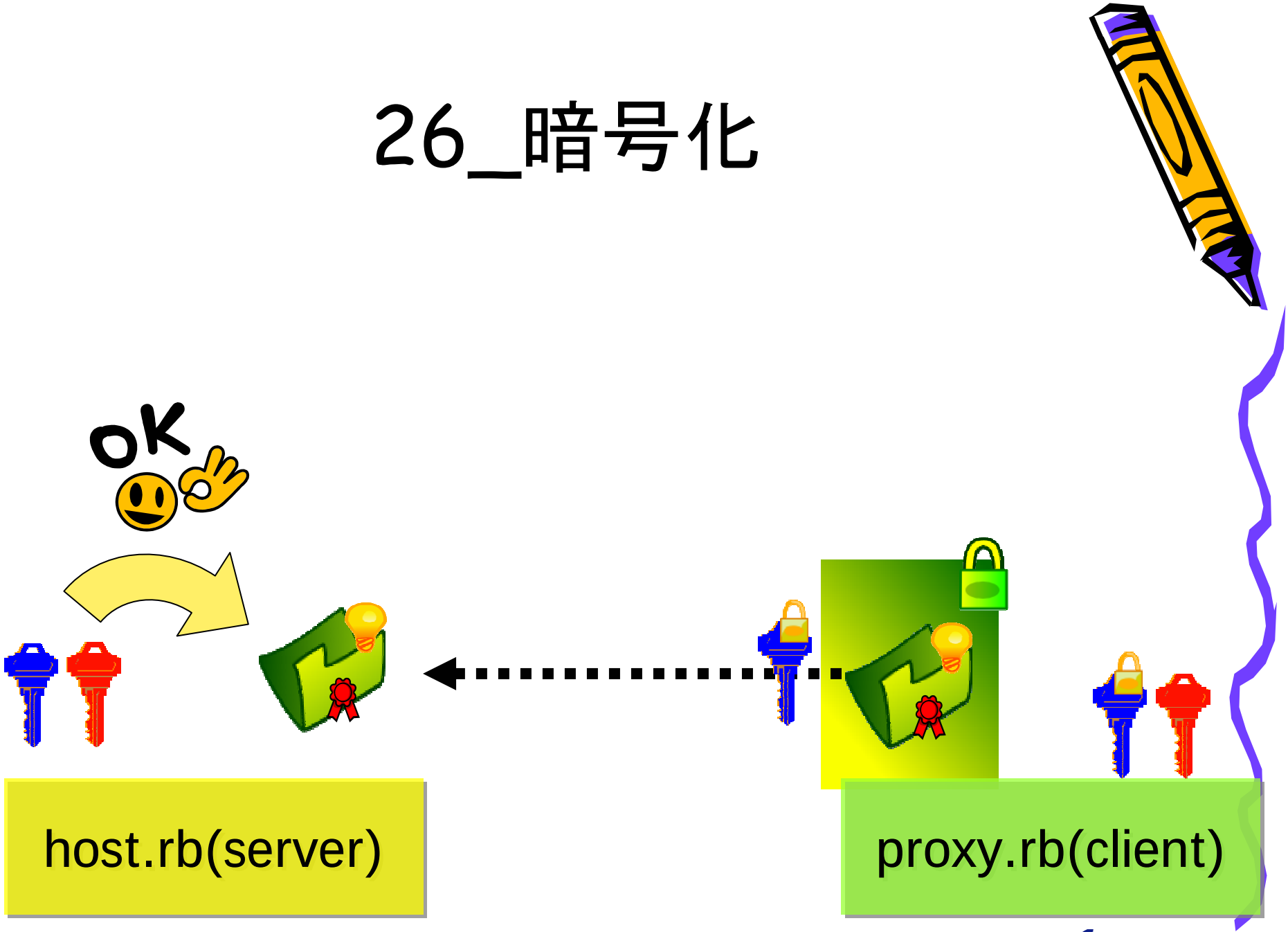


25_リプレー攻撃対策



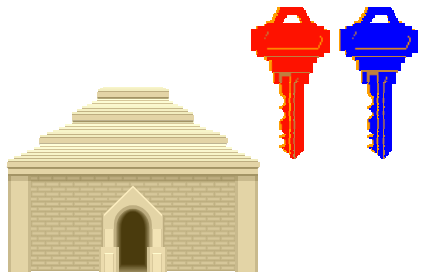
SARION® SYSTEMS RESEARCH

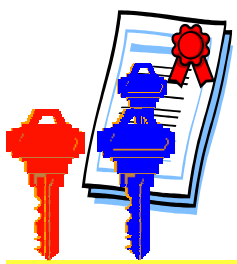
26_暗号化



SARION® SYSTEMS RESEARCH

27_SSL





host.rb(server)

SSL

proxy.rb(client)

SARION® SYSTEMS RESEARCH

3_OpenPGPツールキット

```
pub   4096R/51E96B4B 2006-06-01 [expires: 2011-05-31]
      Key fingerprint = CADB 42B4 3F81 20B6 9C51  8EFD FE5D 20BA 51E9 6B4B
uid                  NAKAMURA, Hiroshi <nakahiro@sarion.co.jp>
sub   2048R/391041A5 2006-06-01 [expires: 2008-05-31]
sub   1024R/13948EAC 2006-06-01 [expires: 2007-06-01]
```

SARION® SYSTEMS RESEARCH

32_OpenPGP署名パケットの生成

```
# 一連の署名手続き: まだまだ生々しい機能しかない…
signature = Signature.new(0x0, 1, 2)
signature.target = plain
signature.secretkey = privkey
signature.hashedsubpacket <<
  SigSubPacket::CreationTime.new(Time.new)
signature.unhashedsubpacket <<
 SigSubPacket::IssuerKeyID.new(pubkey.keyid)
armor = Armor.new(signature.dump)
armor.type = :SIGNATURE
puts armor.dump
```

SARION® SYSTEMS RESEARCH

まとめ

- OpenSSL（ツールキット）は、基本機能が充実している
- しかし結局はバイト配列を扱うC関数群で、扱い辛く、また落とし穴多数。より抽象化したレイヤが必須
- RubyのOpenSSLマッピングモジュールは、それをうまく隠蔽できており、現状でも十分に素晴らしい
- さらにサポートが厚く、必要に応じて機能追加も期待できる
- Rubyで、他の環境では真似のできない、セキュアなアプリケーションを構築しましょう

SARION® SYSTEMS RESEARCH