RUN-TIME ATTACK DETECTION IN CRYPTOGRAPHIC APIS

Marco Squarcina[1], joint work with Riccardo Focardi[1]

August 23, 2017

[1]Università Ca' Foscari Venezia (IT) / Cryptosense (FR)

30th IEEE Computer Security Foundations Symposium (CSF2017)
Santa Barbara (CA), USA

OUTLINE

- Background
- Run-time monitor
- Model
- Analysis
- Implementation
- Conclusions

Background

CRYPTOGRAPHIC API

Host Machine

Crypto API

Trusted Device

CRYPTOGRAPHIC API

Host Machine

Crypto API

Trusted Device

CRYPTOGRAPHIC API

Host Machine

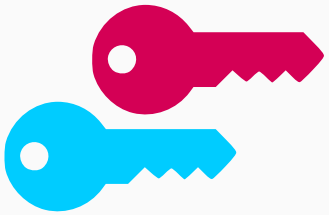
Crypto API

Trusted Device

KEY MANAGEMENT

Host Machine

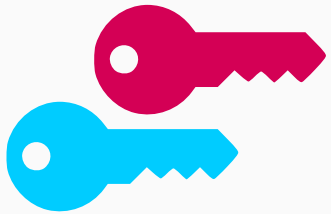
Trusted Device



KEY MANAGEMENT

Host Machine

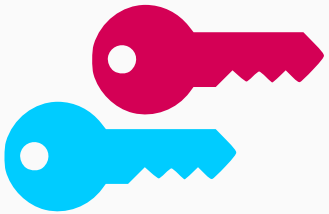
Trusted Device



KEY MANAGEMENT

Host Machine

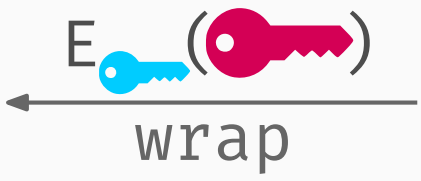
Trusted Device

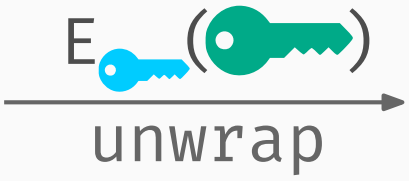


KEY MANAGEMENT

Host Machine

Trusted Device





WRAP/DECRYPT ATTACK [Clu03]

Host Machine

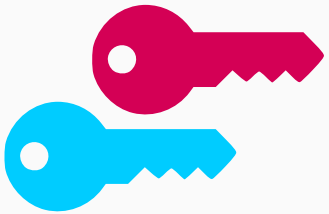
Trusted Device

WRAP/DECRYPT ATTACK [Clu03]

Host Machine

Trusted Device

genkey



WRAP/DECRYPT ATTACK [Clu03]

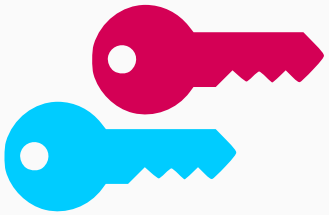
Host Machine

Trusted Device

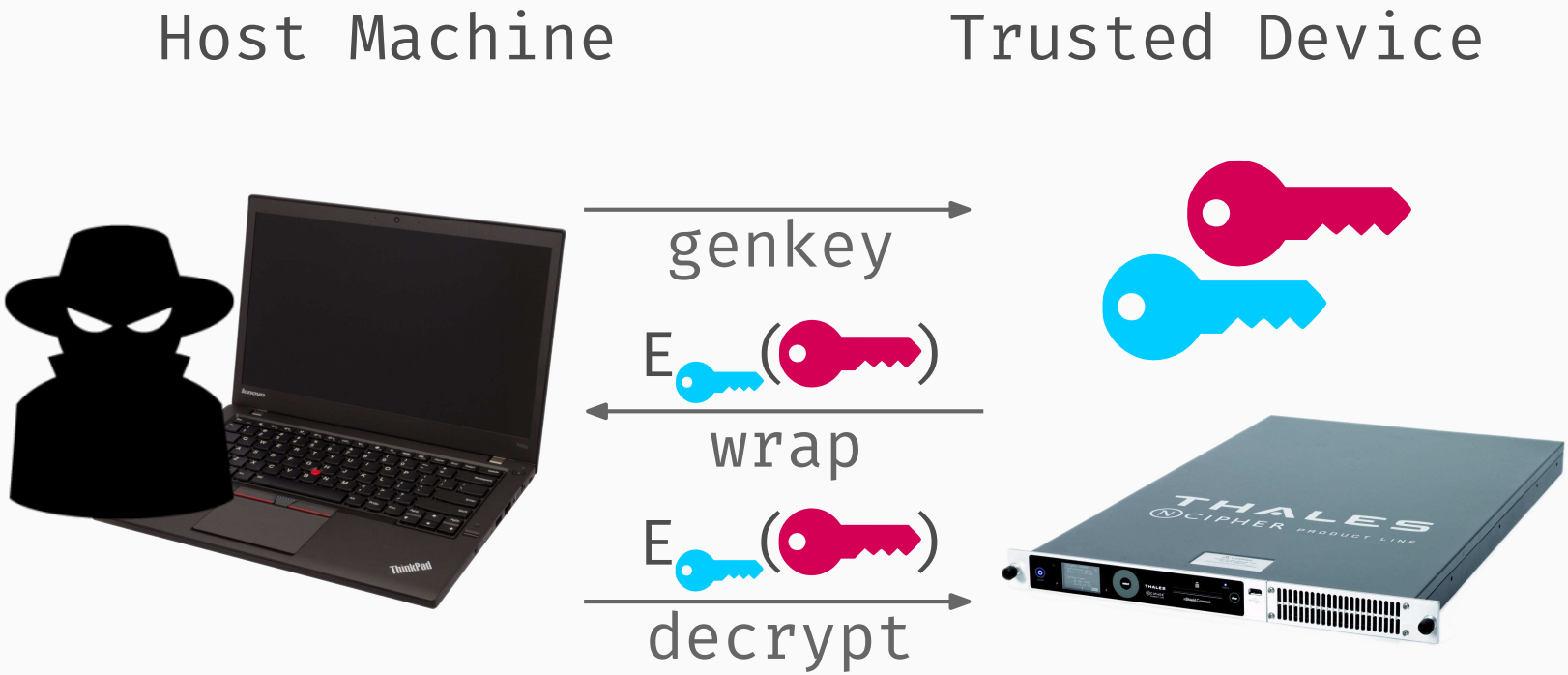
genkey

$E_{\text{key}}(\text{key})$

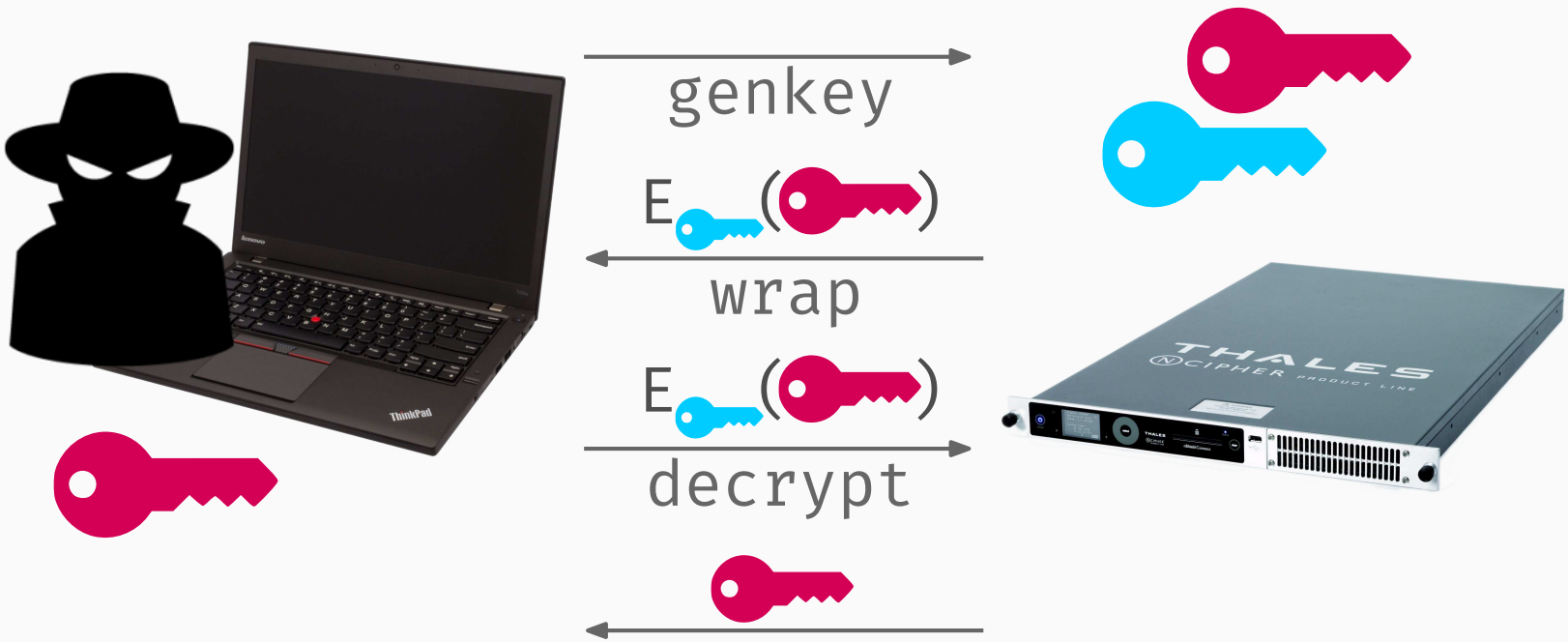
wrap



WRAP/DECRYPT ATTACK [Clu03]



WRAP/DECRYPT ATTACK [Clu03]



SOLUTIONS

Possible solutions

- New cryptographic API [CS09]
- Modifications to current standards [BCFS10]
- Reduction of functionalities

SOLUTIONS

Possible solutions

- New cryptographic API [CS09]
- Modifications to current standards [BCFS10]
- Reduction of functionalities

Difficult to deploy in practice

- Systems are rarely modified
- Legacy applications
- Key management functionalities required

Run-time monitor

A NEW APPROACH

Our proposal

- Collect API invocation sequence for various devices
- Analyse log to detect any leakage of sensitive keys

A NEW APPROACH

Our proposal

- Collect API invocation sequence for various devices
- Analyse log to detect any leakage of sensitive keys

Goals

- Secure
- Accurate
- Distributed
- Efficient

Model

CORE MODEL

Generalisation of the DKS [DKS10] model

- No PKCS#11 specific features (attributes)
- States represent user's knowledge
- Labels on transitions (actions) to capture API calls

Wrap/Decrypt Attack

$$q_0 \xrightarrow{\text{Wrap}(h_{k_1}, h_{k_2})} q_1 \qquad q_1 = q_0 \cup \{E_{k_1}(k_2)\}$$

$$q_1 \xrightarrow{\text{Decrypt}(h_{k_1}, E_{k_1}(k_2))} q_2 \qquad q_2 = q_1 \cup \{k_2\}$$

$$q_0 = \{h_{k_1}, h_{k_2}\}$$

SECURE LOCAL EXECUTION

Dolev-Yao [DY83] model for attacker's deduction capabilities

- Given a set of sensitive keys SK we want to monitor
- Attacker can enc/dec using known keys and keys $\notin SK$
- Executions can include attacker's actions

SECURE LOCAL EXECUTION

Dolev-Yao [DY83] model for attacker's deduction capabilities

- Given a set of sensitive keys SK we want to monitor
- Attacker can enc/dec using known keys and keys $\notin SK$
- Executions can include attacker's actions

Definition (SK-Secure Execution)

An execution σ is secure iff does not leak any of its secure key

$$\sigma = q_0 \xrightarrow{\alpha}\!\!\!\!\rightarrow^{*} q_n \text{ is } SK\text{-secure} \iff SK \cap q_n = \emptyset$$

SECURE EXECUTION

Proposition (characterization of insecure executions)

An execution σ is SK-secure iff none of the following is in σ

- Wrap of a sensitive key under a key not in SK
- Decrypt of a sensitive key encrypted under a sensitive key

SECURE EXECUTION

Proposition (characterization of insecure executions)

An execution σ is SK-secure iff none of the following is in σ

- Wrap of a sensitive key under a key not in SK
- Decrypt of a sensitive key encrypted under a sensitive key

Implications

- Only Wrap and Decrypt API calls must be monitored
- Soundness $\rightarrow$ no false attacks detected
- Completeness $\rightarrow$ all attacks are spotted

SECURE DISTRIBUTED EXECUTION

Definition (Secure Distributed Executions)

$\mathcal{S} = \{(SK_1, \sigma_1), \ldots, (SK_n, \sigma_n)\}$ is a set of distinct executions with their respective sets of sensitive keys.
Let $SK = \bigcup_{i=1,\ldots,n} SK_i$.

$$\mathcal{S} \text{ is secure} \iff \sigma_1, \ldots, \sigma_n \text{ are } SK\text{-secure}$$

SECURE DISTRIBUTED EXECUTION

Definition (Secure Distributed Executions)

$\mathcal{S} = \{(SK_1, \sigma_1), \ldots, (SK_n, \sigma_n)\}$ is a set of distinct executions with their respective sets of sensitive keys.
Let $SK = \bigcup_{i=1,\ldots,n} SK_i$.

$$\mathcal{S} \text{ is secure} \iff \sigma_1, \ldots, \sigma_n \text{ are } SK\text{-secure}$$

Distributed Wrap/Decrypt Attack

$$\sigma = q_0 \xrightarrow{\mathsf{Wrap}(\mathsf{h}_{\mathsf{k}_1}, \mathsf{h}_{\mathsf{k}_2})} q_1 \qquad q_1 = \{\mathsf{h}_{\mathsf{k}_1}, \mathsf{h}_{\mathsf{k}_2}\} \cup \{\mathsf{E}_{\mathsf{k}_1}(\mathsf{k}_2)\}$$

$$\sigma' = q'_0 \xrightarrow{\mathsf{Decrypt}(\mathsf{h}_{\mathsf{k}_1}, \mathsf{E}_{\mathsf{k}_1}(\mathsf{k}_2))} q'_1 \qquad q'_1 = \{\mathsf{h}_{\mathsf{k}_1}, \mathsf{E}_{\mathsf{k}_1}(\mathsf{k}_2)\} \cup \{\mathsf{k}_2\}$$

σ is $\{\mathsf{k}_1, \mathsf{k}_2\}$-secure, σ' is $\{\mathsf{k}_1\}$-secure but not $\{\mathsf{k}_1, \mathsf{k}_2\}$-secure!

Analysis

Distributed Wrap/Decrypt Attack (Partial Execution)

$$\sigma = q_0 \qquad q_0 = \{h_{k_1}, h_{k_2}\}$$

$$\sigma' = q'_0 \xrightarrow{\text{Decrypt}(h_{k_1}, E_{k_1}(k_?))} q'_1 \qquad q'_1 = \{h_{k_1}, E_{k_1}(k_?)\} \cup \{k_?\}$$

LOG ANALYSIS PROBLEM

Distributed Wrap/Decrypt Attack (Partial Execution)

$$\sigma = q_0 \qquad q_0 = \{h_{k_1}, h_{k_2}\}$$

$$\sigma' = q'_0 \xrightarrow{\text{Decrypt}(h_{k_1}, E_{k_1}(k_?))} q'_1 \qquad q'_1 = \{h_{k_1}, E_{k_1}(k_?)\} \cup \{k_?\}$$

$k_? = k_2$ is leaked but cannot be linked to one of the handles!

LOG ANALYSIS PROBLEM

Distributed Wrap/Decrypt Attack (Partial Execution)

$$\sigma = q_0 \qquad q_0 = \{h_{k_1}, h_{k_2}\}$$

$$\sigma' = q'_0 \xrightarrow{\text{Decrypt}(h_{k_1}, E_{k_1}(k_?))} q'_1 \qquad q'_1 = \{h_{k_1}, E_{k_1}(k_?)\} \cup \{k_?\}$$

$k_? = k_2$ is leaked but cannot be linked to one of the handles!

Key Fingerprint

- Terms can only be compared by syntactic equality
- Enrich logs with a special one-way deterministic function
- $h_y \xrightarrow{\text{KeyFprint}} \text{kf}(y),\ y' \to \text{kf}(y'),\ y = y' \iff \text{kf}(y) = \text{kf}(y')$

LOG ANALYSIS USING KEY FINGERPRINTING

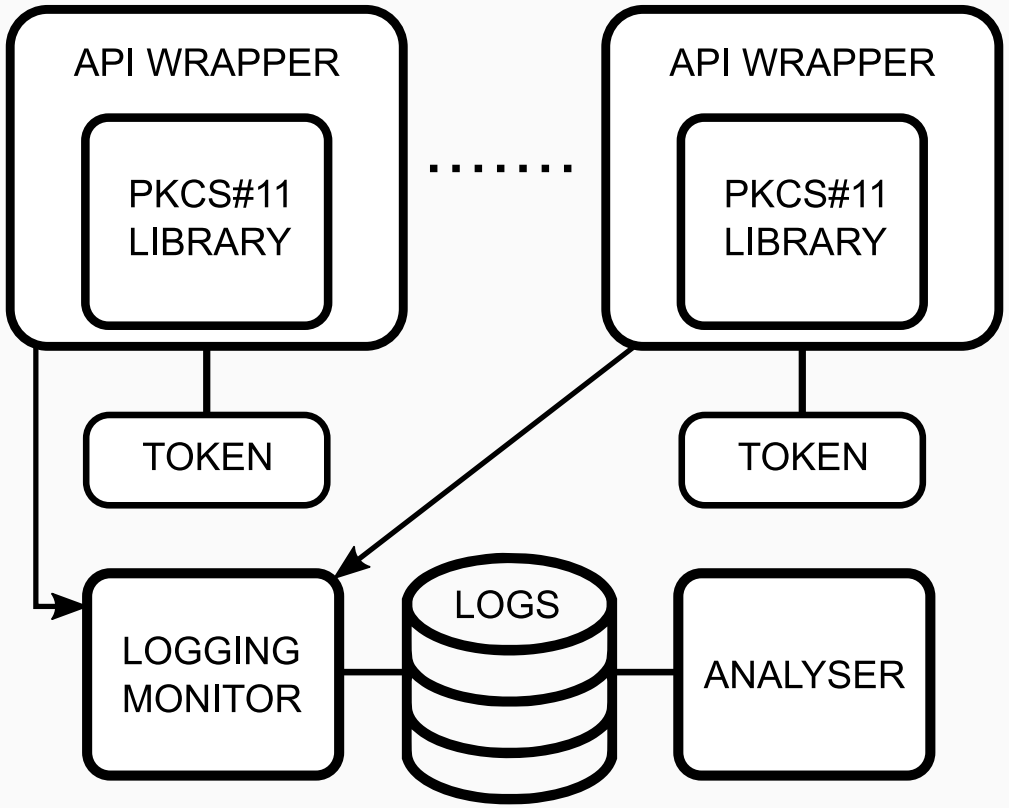
Given logs and handles of sensitive keys:

1. Collect all the fingerprints of sensitive keys
2. For each wrap call
   - if a sensitive key is wrapped under an insecure one → ATTACK
3. For each decrypt call
   - if the decryption key is sensitive
   - compute the fingerprint of the result and compare it against the set of fingerprints collected at step 1
   - if a match is found → ATTACK

Implementation

LOG ANALYSIS TOOL FOR PKCS#11

The tool is able to detect all the key-management attacks found in the literature [DKS10, FLS10] involving symmetric encryption operations



LOG ANALYSIS TOOL FOR PKCS#11

Instrumented API functions

- `C_WrapKey`
- `C_Decrypt`
- `C_GetAttributeValue`
- `C_GenerateKey`
- `C_Login`

LOG ANALYSIS TOOL FOR PKCS#11

Instrumented API functions

- `C_WrapKey`
- `C_Decrypt`
- `C_GetAttributeValue`
- `C_GenerateKey`
- `C_Login`

Possible fingerprints for a key depending on its attributes

- encrypt $\rightarrow \mathrm{kf}(\mathrm{k})_E = \langle \mathrm{r}, \mathrm{E}_\mathrm{k}(\mathrm{r}) \rangle$
- decrypt $\rightarrow \mathrm{kf}(\mathrm{k})_D = \langle \mathrm{r}, \mathrm{D}_\mathrm{k}(\mathrm{r}) \rangle$
- wrap $\rightarrow \mathrm{kf}(\mathrm{k})_W = \langle \mathrm{E}_\mathrm{k}(\mathrm{k}) \rangle$

Conclusions

CONTRIBUTIONS

- Provided a model for distributed run-time detection of crypto APIs attacks
- Devised a sound and complete characterization of attacks
- Proved that the problem of offline attack detection is unsolvable
- ...but key fingerprinting mechanism enables feasible and efficient analysis
- Developed a proof-of-concept log analysis tool for PKCS#11

FUTURE WORKS

- Reason about practical implementations of key fingerprint
- Cover a more extensive fragment of PKCS#11 with the tool and implement a key fingerprint call the API using software emulators
- Characterize other crypto APIs and study formally which are the problematic rules that should be tracked in the logs
- Formally devise a logging policy to prevent logs to grow indefinitely

REFERENCES

[BCFS10] M. Bortolozzo, M. Centenaro, R. Focardi, and G. Steel. **Attacking and fixing PKCS#11 security tokens.** In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS'10)*, pages 260–269, Chicago, Illinois, USA, October 2010. ACM Press.

[Clu03] J. Clulow. **On the security of PKCS#11.** In *Proceedings of the 5th International Workshop on Cryptographic Hardware and Embedded Systems (CHES'03)*, volume 2779 of *LNCS*, pages 411–425. Springer, 2003.

[CS09] V. Cortier and G. Steel. **A generic security API for symmetric key management on cryptographic devices.** In Michael Backes and Peng Ning, editors, *Proceedings of the 14th European Symposium on Research in Computer Security (ESORICS'09)*, volume 5789 of *Lecture Notes in Computer Science*, pages 605–620, Saint Malo, France, September 2009. Springer.

[DKS10] S. Delaune, S. Kremer, and G. Steel. **Formal analysis of PKCS#11 and proprietary extensions.** *Journal of Computer Security*, 18(6):1211–1245, November 2010.

[DY83] D. Dolev and A. Yao. **On the security of public key protocols.** *IEEE Transactions in Information Theory*, 2(29):198–208, March 1983.

[FLS10] R. Focardi, F.L. Luccio, and G. Steel. **An introduction to security api analysis.** In *FOSAD*, pages 35–65, 2010.

Thank you!

Questions?

Bonus Slides

Algorithm 1 Log Analysis using Key Fingerprinting

```
1:  procedure LogAnalysis(σ̄, H)
2:      FSK = [ ]
3:      for (a, ret) ∈ σ̄ do
4:          if a == KeyFprint(h) and h ∈ H then
5:              FSK ← FSK + [ret]
6:          end if
7:      end for
8:      for (a, ret) ∈ σ̄ do
9:          if a == Wrap_*(h_1, h_2) and h_1 ∉ H and h_2 ∈ H then
10:             return a
11:         end if
12:         if a == Decrypt_*(h, t) and h ∈ H and kf(ret) ∈ FSK then
13:             return a
14:         end if
15:     end for
16:     return None
17: end procedure
```

API RULES

$$\xrightarrow[\text{new } n, k]{\text{KeyGen}} \quad h_k$$

$$\xrightarrow[\text{new } n, s]{\text{KeyPairGen}} \quad h_{\text{priv}(s)},\ \text{pub}(s)$$

$$h_{y_1},\ h_{y_2} \xrightarrow{\text{Wrap}} E_{y_1}(y_2)$$

$$h_{\text{priv}(z)},\ h_{y_2} \xrightarrow{\text{Wrap}_{sa}} \text{aenc}(y_2,\ \text{pub}(z))$$

$$h_{y_1},\ h_{\text{priv}(z)} \xrightarrow{\text{Wrap}_{as}} E_{y_1}(\text{priv}(z))$$

$$h_{y_2}, E_{y_2}(y_1) \xrightarrow[\text{new } n_1]{\text{Unwrap}} h_{y_1}$$

$$h_{\text{priv}(z)}, \text{aenc}(y_1,\ \text{pub}(z)) \xrightarrow[\text{new } n_1]{\text{Unwrap}_{sa}} h_{y_1}$$

$$h_{y_2}, E_{y_2}(\text{priv}(z)) \xrightarrow[\text{new } n_1]{\text{Unwrap}_{as}} h_{\text{priv}(z)}$$

$$h_{y_1}, y_2 \xrightarrow{\text{Encrypt}} E_{y_1}(y_2)$$

$$h_{y_1}, E_{y_1}(y_2) \xrightarrow{\text{Decrypt}} y_2$$

$$h_{\text{priv}(z)}, y_1 \xrightarrow{\text{Encrypt}_a} \text{aenc}(y_1,\ \text{pub}(z))$$

$$h_{\text{priv}(z)}, \text{aenc}(y_2,\ \text{pub}(z)) \xrightarrow{\text{Decrypt}_a} y_2$$