Formal Analysis of Key Management APIs

Graham Steel

INRIA & LSV, ENS de Cachan
Cryptographic key management

The ‘elephant in the room’ of cryptographic security

- Key creation and destruction
- Key establishment and distribution
- Key storage and backup
- Key use according to policy

For many hundreds of keys (every employee laptop, smartcard, credential, ticket, token, device, ...)

.. and all in a secure, robust way in a distributed system in a hostile environment
Crypto Basics

We consider only symmetric key crypto

Problem is now the security of key $K$
Model

Signature $\Sigma ::= N, X, F, P$

Plain terms

$\begin{align*}
  t, t_i & ::= x \quad x \in X \\
         & | n \quad n \in N \\
         & | f(t_1, \ldots, t_n) \quad f \in F
\end{align*}$

Facts

$l = \{ p(t, b) \mid p \in P, t \in T, b \in \{ \top, \bot \} \}$

Rules

$T; L \xrightarrow{\text{new} \tilde{n}} T'; L'$
HSMs

- Manufacturers include IBM, VISA, nCipher, Thales, Utimaco, HP
- Cost around $10 000
A Word About Your PIN

IPIN derived by:

Write account number (PAN) as 0000ABCDEFGHIJKLMNOPQRSTUVWXYZ

Encrypt under a PIN Derivation Key (PDK)

\[ \{PAN\}_{PDK} = IPIN \]

PIN = IPIN + Offset (modulo 10 each digit)

Offset NOT secure!
Master Key Scheme

Host machine

\{ \text{TMK1} \}_{\text{KM}}

\{ \text{PDK1} \}_{\text{KM}}

HSM

KM

TMK = Terminal Master Key
Example: Send PDK to Terminal

\[
\{\text{PDK1}\}_{\text{km}}, \{\text{TMK1}\}_{\text{km}} \rightarrow \{\text{PDK1}\}_{\text{TMK1}}
\]
Terminal Comms (TC) Key
Managing Key Types

Host machine

\{ \text{TMK1} \} \quad \text{KM}

\{ \text{PDK1} \} \quad \text{KM}

\{ \text{TC1} \} \quad \text{KM2}

\text{VSM}

\text{KM}

\text{KM2}
Example: Enter TC key

\[ TC \rightarrow \{TC\}_{km2} \]
Example: Send TC to Terminal

\[
\begin{align*}
\{TC1\}_{KM2} & \rightarrow \text{KM, KM2} \\
\text{KM} & \rightarrow \{TC1\}_{TMK1} \\
\{TC\}_{km2}, \{TMK1\}_{km} & \rightarrow \{TC\}_{TMK1}
\end{align*}
\]
Attack - Step 1

PAN $\rightarrow$ KM, KM2

VSM

PAN $\rightarrow$ \{PAN\} km2

\{PAN\} KM2
Attack - Step 2

\[
\begin{align*}
\{PAN\}_{KM2} & \rightarrow \text{KM, KM2} \\
\{PAN\}_{KM2}, \{PDK1\}_k & \rightarrow \{PAN\}_{PDK1}
\end{align*}
\]
VSM - Formal Model

\[ X, Y \rightarrow \{X\}_Y \]
\[ \{X\}_Y, Y \rightarrow X \]
\[ \mathit{new\ tmk} \rightarrow \{\mathit{tmk}\}^{\mathit{km}} \]
\[ \mathit{TC} \rightarrow \{\mathit{TC}\}^{\mathit{km2}} \]
\[ \{\mathit{PDK}\}^{\mathit{km}}, \{\mathit{TMK}\}^{\mathit{km}} \rightarrow \{\mathit{PDK}\}^{\mathit{TMK}} \]
\[ \{\mathit{TC}\}^{\mathit{km2}}, \{\mathit{TMK}\}^{\mathit{km}} \rightarrow \{\mathit{TC}\}^{\mathit{TMK}} \]

\[ I = \{\{\mathit{pdk}\}^{\mathit{km}}, \mathit{pan}\}, +8 \text{ more rules} \]

SWV237, 238 (www.tptp.org)

CASC at FLoC ’10: 9/17 provers can find the attack, only E can find model
PKCS #11

Host machine

- n1
- n2

Trusted device

- k1
- k2

A(n1)

A(n2)
PKCS#11

Ubiquitous in authentication tokens, smartcards, ...

RSA PKCS#11 is specified in a 400 page document

We consider here a core fragment of key management operations

Not included: signing, verification, certificate management, etc.

\[ \text{h}(n_1, k_1) \] - a handle \( n_1 \) for key \( k_1 \) (\( h \) is a \textit{private symbol})

\[ a_1(n_1) \] - setting of attribute \( a_1 \) for handle \( n_1 \)

We consider attributes:

\text{encrypt}(n), \text{decrypt}(n), \text{ sensitive}(n)

\text{extract}(n), \text{wrap}(n), \text{unwrap}(n)
Key Management - 1

KeyGenerate:

\[ \text{new } n, k \rightarrow h(n, k); L \]

Where \( L = \neg \text{extractable}(n), \neg \text{wrap}(n), \neg \text{unwrap}(n), \]
\( \neg \text{encrypt}(n), \neg \text{decrypt}(n), \neg \text{sensitive}(n) \)
Set_Wrap : \( h(x_1, y_1); \neg \text{wrap}(x_1) \rightarrow ;\text{wrap}(x_1) \)

Set_Encrypt : \( h(x_1, y_1); \neg \text{encrypt}(x_1) \rightarrow ;\text{encrypt}(x_1) \)

\[ \vdots \]

UnSet_Wrap : \( h(x_1, y_1); \text{wrap}(x_1) \rightarrow ;\neg \text{wrap}(x_1) \)

UnSet_Encrypt : \( h(x_1, y_1); \text{encrypt}(x_1) \rightarrow ;\neg \text{encrypt}(x_1) \)

\[ \vdots \]

Some restrictions, e.g. can’t unset sensitive
Wrap:
\[ h(x_1, y_1), h(x_2, y_2); \text{wrap}(x_1), \quad \rightarrow \quad \{y_2\}_{y_1} \]
\[ \text{extract}(x_2) \]

Unwrap:
\[ h(x_2, y_2), \{y_1\}_{y_2}; \text{unwrap}(x_2) \xrightarrow{\text{new } n_1} h(n_1, y_1); \text{extract}(n_1), L \]

where \( L = \)
\[ \neg\text{wrap}(n_1), \neg\text{unwrap}(n_1), \neg\text{encrypt}(n_1), \neg\text{decrypt}(n_1), \neg\text{sensitive}(n_1). \]
Key Usage

Encrypt:
\[ h(x_1, y_1), y_2; \text{encrypt}(x_1) \rightarrow \{y_2\}_{y_1} \]

Decrypt:
\[ h(x_1, y_1), \{y_2\}_{y_1}; \text{decrypt}(x_1) \rightarrow y_2 \]
Key Separation Attack (Clulow, 2003)

**Intruder knows:** $h(n_1, k_1), h(n_2, k_2)$.

**State:** \( \text{wrap}(n_2), \text{decrypt}(n_2), \text{sensitive}(n_1), \text{extract}(n_1) \)

**Wrap:** $h(n_2, k_2), h(n_1, k_1) \rightarrow \{k_1\}_{k_2}$

**Decrypt:** $h(n_2, k_2), \{k_1\}_{k_2} \rightarrow k_1$
Host machine

n1

\{k1\}k2

k1

Trusted device

k1

x,s

k2

w,d

PKCS #11
Fix decrypt/wrap attack..

Set_Wrap : \( h(x_1, y_1); \neg \text{wrap}(x_1), \neg \text{decrypt}(x_1) \rightarrow \text{wrap}(x_1) \)

Set_Decrypt : \( h(x_1, y_1); \neg \text{wrap}(x_1), \neg \text{decrypt}(x_1) \rightarrow \text{decrypt}(x_1) \)

Unset_Wrap

Unset_Decrypt
Another Attack

**Intruder knows:** $h(n_1, k_1)$, $h(n_2, k_2)$, $k_3$

**State:** sensitive($n_1$), extract($n_1$), unwrap($n_2$), encrypt($n_2$)

- **SEncrypt:** $h(n_2, k_2)$, $k_3$ $\rightarrow$ $\{k_3\}_{k_2}$
- **Unwrap:** $h(n_2, k_2)$, $\{k_3\}_{k_2}$ $\xrightarrow{\text{new } n_3}$ $h(n_3, k_3)$
- **Set-wrap:** $h(n_3, k_3)$ $\rightarrow$ wrap($n_3$)
- **Wrap:** $h(n_3, k_3)$, $h(n_1, k_1)$ $\rightarrow$ $\{k_1\}_{k_3}$
- **Intruder:** $\{k_1\}_{k_3}$, $k_3$ $\rightarrow$ $k_1$
Fix decrypt/wrap, encrypt/unwrap..

Intruder knows: $h(n_1, k_1), h(n_2, k_2), k_3$

State: sensitive($n_1$), extract($n_1$), extract($n_2$)

Set wrap: $h(n_2, k_2) \rightarrow \text{;wrap}(n_2)$

Set wrap: $h(n_1, k_1) \rightarrow \text{;wrap}(n_1)$

Wrap: $h(n_1, k_1), h(n_2, k_2) \rightarrow \{k_2\}_{k_1}$

Set unwrap: $h(n_1, k_1) \rightarrow \text{;unwrap}(n_1)$

Unwrap: $h(n_1, k_1), \{k_2\}_{k_1} \xrightarrow{\text{new } n_3} h(n_3, k_2)$

Wrap: $h(n_2, k_2), h(n_1, k_1) \rightarrow \{k_1\}_{k_2}$

Set decrypt: $h(n_3, k_2) \rightarrow \text{;decrypt}(n_3)$

Decrypt: $h(n_3, k_2), \{k_1\}_{k_2} \rightarrow k_1$
PKCS #11
Modes

\[ h : \text{Nonce} \times \text{Key} \to \text{Handle} \]

\[ \text{senc} : \text{Key} \times \text{Key} \to \text{Cipher} \]

\[ \text{aenc} : \text{Key} \times \text{Key} \to \text{Cipher} \]

\[ \text{pub} : \text{Seed} \to \text{Key} \]

\[ \text{priv} : \text{Seed} \to \text{Key} \]

\[ a : \text{Nonce} \to \text{Attribute} \quad \text{for all } a \in \mathcal{A} \]

\[ x_1, x_2, n_1, n_2 : \text{Nonce} \]

\[ y_1, y_2, k_1, k_2 : \text{Key} \]

\[ z, s : \text{Seed} \]

See Delaune, Kremer & S., *Formal Analysis of PKCS#11*, CSF ’08
Two kinds of problem

- A bad ‘attribute policy’
  
  One can set conflicting attributes for a key

- Policy not enforced
  
  By copying the key using wrap/unwrap, can ‘escape’ the policy

Attack this problem by first formalising ‘attribute policy’
KeyGenerate: \[\text{new } n_1,k_1 \rightarrow h(n_1,k_1); L(n_1), \neg \text{extract}(n_1)\]

Wrap:
\[h(x_1,y_1), h(x_2,y_2); \text{wrap}(x_1), \text{extract}(x_2) \rightarrow \{y_2\}_{y_1}\]

Unwrap:
\[h(x_2,y_2), \{y_1\}_{y_2}; \text{unwrap}(x_2) \xrightarrow{\text{new } n_1} h(n_1,y_1); L(n_1)\]

Encrypt:
\[h(x_1,y_1), y_2; \text{encrypt}(x_1) \rightarrow \{y_2\}_{y_1}\]

Decrypt:
\[h(x_1,y_1), \{y_2\}_{y_1}; \text{decrypt}(x_1) \rightarrow y_2\]

Set_Encrypt:
\[h(x_1,y_1); \neg \text{encrypt}(x_1) \rightarrow \text{encrypt}(x_1)\]

UnSet_Encrypt:
\[h(x_1,y_1); \text{encrypt}(x_1) \rightarrow \neg \text{encrypt}(x_1)\]

\[\vdots\]
KeyGenerate: \[ \text{new } n_1, k_1 \rightarrow h(n_1, k_1); A(n_1) \]

Wrap:
\[ h(x_1, y_1), h(x_2, y_2); \text{wrap}(x_1), \text{extract}(x_2) \rightarrow \{y_2\}_{y_1} \]

Unwrap:
\[ h(x_2, y_2), \{y_1\}_{y_2}; \text{unwrap}(x_2) \xrightarrow{\text{new } n_1} h(n_1, y_1); A(n_1) \]

Encrypt:
\[ h(x_1, y_1), y_2; \text{encrypt}(x_1) \rightarrow \{y_2\}_{y_1} \]

Decrypt:
\[ h(x_1, y_1), \{y_2\}_{y_1}; \text{decrypt}(x_1) \rightarrow y_2 \]

Set_Attribute_Value:
\[ h(x_1, y_1); A_1(x_1) \rightarrow A_2(x_1) \]
Attribute Policy

An *attribute policy* is a finite directed graph $P = (S\_P, \rightarrow\_P)$ where $S\_P$ is the set of allowable object states, and $\rightarrow\_P \subseteq S\_P \times S\_P$ is the set of allowable transitions between the object states.

An attribute policy $P = (S, \rightarrow)$ is *complete* if $P$ consists of a collection of disjoint, disconnected cliques, and for each clique $C$, $c_0, c_1 \in C \Rightarrow c_0 \cup c_1 \in C$

We insist on complete policies, assuming intruder can always copy keys.
Endpoints

We call the object states of $S$ that are maximal in $S$ with respect to set inclusion *end points* of $P$.

Theorem: Derivation in API with complete policy iff derivation in API with (static) endpoint policy
Bounds

Assume endpoint policies

Make series of simple transformations

- Bound number of fresh keys to number of endpoints \( \# \text{ep} \)
  - get the same key every time a particular endpoint is requested

- Bound number of handles to \((\# \text{ep})^2\)
  - for each key, get one handle for each endpoint

Intruder always starts with his own key

so require \( \# \text{ep} + 1 \) keys and \((\# \text{ep} + 1)^2\) handles
KeyGenerate: \[ \text{new } n_1,k_1 \rightarrow h(n_1,k_1); A(n_1) \]

Wrap:
\[ h(x_1,y_1), h(x_2,y_2); \text{wrap}(x_1), A(x_2) \xrightarrow{\text{new } m_k} \{y_2\}_{y_1}, \{m_k\}_{y_1} \]
\[ \text{hmac}_{m_k}(y_2, A) \]

Unwrap:
\[ h(x_2,y_2), \{y_1\}_{y_2}, \{x_m\}_{y_2}, \xrightarrow{\text{new } n_1} h(n_1,y_1); A(n_1) \]
\[ \text{hmac}_{x_m}(y_1, A); \text{unwrap}(x_2) \]

Encrypt:
\[ h(x_1,y_1), y_2; \text{encrypt}(x_1) \rightarrow \{y_2\}_{y_1} \]

Decrypt:
\[ h(x_1,y_1), \{y_2\}_{y_1}; \text{decrypt}(x_1) \rightarrow y_2 \]

\[ P = (\{e,d,ed,w,u,wu\}, \rightarrow) \text{ (where } \rightarrow \text{ makes the obvious cliques)} \]
Model checking

Use SATMC (U. di Genova) to check formal model for attack

A *known key* is a key $k$ such that the intruder knows the plaintext value $k$ and the intruder has a handle $h(n, k)$.

**Property 1** If an intruder starts with no known keys, he cannot obtain any known keys.

Verified for our revised API in 0.4 sec

**Property 2** If an intruder starts with a known key $k_i$ with handle $h(n_i, k_i)$, and $ed(n_i)$ is true, then he cannot obtain any further known keys.

Attack!
Lost session key attack

**Initial knowledge:** Handles $h(n_1, k_1)$, $h(n_2, k_2)$, and $h(n_i, k_i)$. Key $k_i$. Attributes $ed(n_1)$, $wu(n_2)$, $ed(n_i)$.

**Trace:**

Wrap: (ed) $h(n_2, k_2), h(n_i, k_i) \rightarrow$

\{k_i\}_k_2, \{k_3\}_k_2, hmac_{k_3}(k_i, ed)$

Unwrap: (wu) $h(n_2, k_2), \{k_i\}_k_2, \{k_i\}_k_2,$

$hmac_{k_i}(k_i, wu) \rightarrow h(n_2, k_i)$

Wrap: (ed) $h(n_2, k_i), h(n_1, k_1) \rightarrow$

\{k_1\}_k_i, \{k_3\}_k_i, hmac_{k_3}(k_1, ed)$

Decrypt: $k_i, \{k_1\}_k_i \rightarrow k_1$
Revised API

Wrap:
\[ h(x_1, y_1), h(x_2, y_2); \text{wrap}(x_1), A(x_2) \xrightarrow{\text{new } m_k} \{y_2\}y_1, \{m_k\}_y_1 \text{ hmac}_{m_k}(y_2, A, y_1) \]

Unwrap:
\[ h(x_2, y_2), \{y_1\}_y_2, \{x_m\}_y_2, \text{ hmac}_{x_m}(y_1, A, y_2); \text{unwrap}(x_2) \xrightarrow{\text{new } n_1} h(n_1, y_1); A(n_1) \]

Property 2 now verified by SATMC

Can also verify attribute policy is enforced
<table>
<thead>
<tr>
<th>Device</th>
<th>Brand</th>
<th>Model</th>
<th>sym</th>
<th>asym</th>
<th>cobj</th>
<th>chan</th>
<th>w</th>
<th>ws</th>
<th>a1</th>
<th>a2</th>
<th>a3</th>
<th>a4</th>
<th>a5</th>
<th>mc</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB</td>
<td>XXXX</td>
<td>XXXX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>XXXX</td>
<td>XXXX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>a1</td>
</tr>
<tr>
<td></td>
<td>XXXX</td>
<td>XXXX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>XXXX</td>
<td>XXXX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>XXXX</td>
<td>XXXX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>XXXX</td>
<td>XXXX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>XXXX</td>
<td>XXXX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>XXXX</td>
<td>XXXX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Card</td>
<td>XXXX</td>
<td>XXXX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>XXXX</td>
<td>XXXX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>XXXX</td>
<td>XXXX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>XXXX</td>
<td>XXXX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>XXXX</td>
<td>XXXX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>XXXX</td>
<td>XXXX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Soft</td>
<td>XXXX</td>
<td>XXXX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>XXXX</td>
<td>XXXX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
A New Hope?

Proposals for new APIs by Cachin and Chandran (CSF ’09), Cortier and Steel (ESORICS ’09).

– CC is for a single central server with a log, CS is for distributed tokens
– possibility of unifying these proposals?

Standards processes trying to set new APIs

– OASIS Key Management Interoperability Protocol
– IEEE Security in Storage Working Group
– PKCS#11 2.30 (no improvement)
Cachin-Chandran API

- Assume only one key server, many users, log of all operations

- Keys created with no attributes. Owner of key can set permissions

- Conflicts are checked by looking in the log, e.g. ’if this key has been used by any user for wrapping, do not allow it to be used for decryption’

- Also calculates dependencies between keys

+ very flexible, - fails immediately if a key is compromised, or if distributed over several servers
Cortier-Steel API

- Assume distributed tokens, one for each user
- Strict hierarchy of wrap/unwrap and encrypt/decrypt keys
- Keys created with attributes that cannot be changed in future
- Key attributes include names of other users key can be shared with
- All encryptions tagged with key/user information

+ strong security properties, robust to loss of keys, no central log required

- not as flexible as Cachin proposal
More on Key Management APIs


V. Cortier and G. Steel. *A Generic API for Symmetric Key Management*. In ESORICS ’09.


S. Fröschle and G. Steel. *Analysis of PKCS#11 APIs with Unbounded Fresh Data*, ARSPA-WITS ’09.

OASIS [www.oasis-open.org/committees/kmip](http://www.oasis-open.org/committees/kmip), IEEE 1619

siswg.net

ASA-4, [http://www.lsv.ens-cachan.fr/~steel/asa4](http://www.lsv.ens-cachan.fr/~steel/asa4)

Interested? Internships + postdocs available, get in touch