Using Types in Composed Cryptographic Proofs

EUTypes 2018

Konrad Kohbrok

October 9, 2018

Department of Mathematics and Systems Analysis, Aalto University
Joint work with Microsoft Research Cambridge and
the University of Edinburgh

Quick Primer: miTLS

- Formally verified implementation of TLS

Quick Primer: miTLS

- Formally verified implementation of TLS
    - memory safety
    - functional correctness
    - cryptographic verification

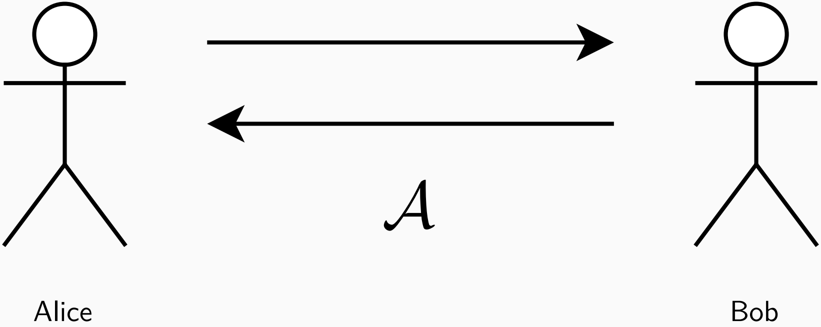
Quick Primer: miTLS

- Formally verified implementation of TLS
  - memory safety
  - functional correctness
  - cryptographic verification
- Not widely understood by cryptographers

What is Security?

Example: Authenticated Encryption

Alice and Bob expect confidentiality and authenticity and use some authenticated encryption scheme $\sigma = \{\mathsf{Enc}, \mathsf{Dec}\}$.



Real- vs Ideal Behaviour

We define Security in terms of Real- vs. Ideal behaviour of the authenticated encryption scheme σ.

Real Behaviour

Alice		**Bob**
m, k		k
$c \leftarrow\!\!\$\ \sigma.\mathrm{Enc}(k, m)$	$\xrightarrow{\quad c \quad}$	
		$m \leftarrow \sigma.\mathrm{Dec}(k, c)$

Ideal Behaviour

Ideal Behaviour

Alice		**Bob**
m, k		k
$c \leftarrow_{\$} \{0,1\}^{\|m\|}$		
$M[c] \leftarrow m$	$\xrightarrow{c}$	
		$m \leftarrow M[c]$

Security Game

Alice and Bob as Oracles

We turn Alice (encryption) and Bob (decryption) into oracles and bundle them into a package we call Authenticated Encryption (AE).

$$\mathcal{A} \xrightarrow{\mathsf{ENC}, \mathsf{DEC}} \boxed{\mathtt{AE}_\sigma^b}$$

Bit b determines if $\mathtt{AE}$ exhibits real $(b = 0)$ or ideal behaviour $(b = 1)$ of scheme σ.

Computational Indistinguishability

σ is AE-secure if for all efficient adversaries (i.e., algorithms) $\mathcal{A}$, we have that

$$\boxed{\mathtt{AE}_{\sigma}^{0}} \quad \overset{\mathcal{A}}{\approx} \quad \boxed{\mathtt{AE}_{\sigma}^{1}}$$

How to Prove this?

Building Blocks

Commonly, authenticated encryption schemes are built from two components:

- one providing confidentiality (encryption scheme ϵ)
- one providing authenticity (mac scheme μ)

$$
\begin{array}{l}
\underline{\sigma.\mathsf{Enc}(k, m)\qquad\qquad} \\
k_1 || k_2 \leftarrow k \\
c \leftarrow \epsilon.\mathsf{Enc}(k1, m) \\
tag \leftarrow \mu.\mathsf{MAC}(k2, c) \\
\textbf{return } (tag || c)
\end{array}
$$

Assumptions

One assumption for each component:

Assumption 1: Encryption scheme ϵ provides confidentiality

Assumption 2: MAC scheme μ provides authenticity

$$\forall \mathcal{A} : \texttt{Conf.}^0_\epsilon \overset{\mathcal{A}}{\approx} \texttt{Conf.}^1_\epsilon$$

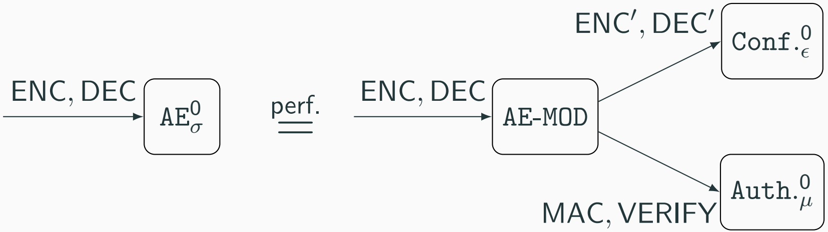
$$\forall \mathcal{A} : \texttt{Auth.}^0_\mu \overset{\mathcal{A}}{\approx} \texttt{Auth.}^1_\mu$$

Reductions

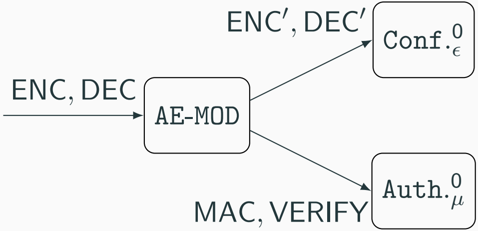
- Goal: Show that $\forall \mathcal{A} : \mathtt{AE}^0_\sigma \overset{\mathcal{A}}{\approx} \mathtt{AE}^1_\sigma$.

Reductions

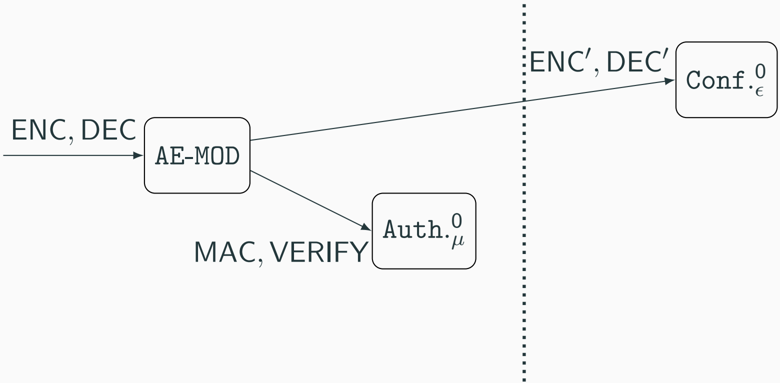
- Goal: Show that $\forall \mathcal{A} : \texttt{AE}^0_\sigma \overset{\mathcal{A}}{\approx} \texttt{AE}^1_\sigma$.



Proof

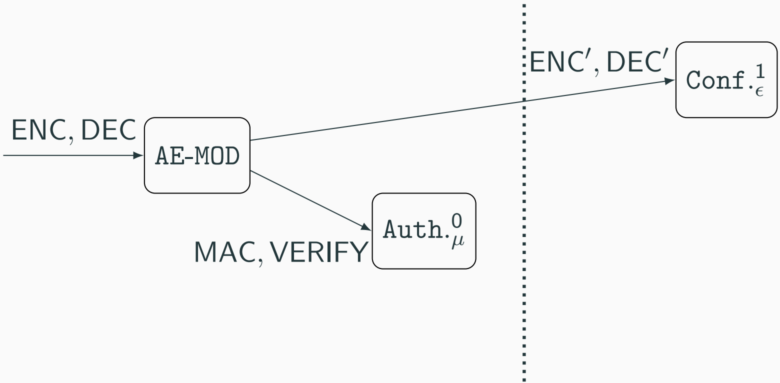


Proof

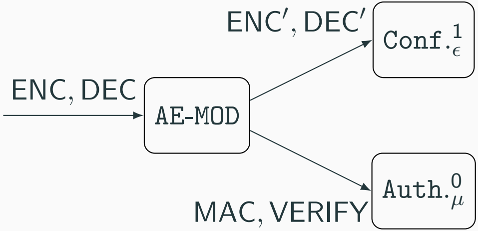


ENC, DEC
AE-MOD
ENC′, DEC′
$\mathtt{Conf.}^{0}_{\epsilon}$
MAC, VERIFY
$\mathtt{Auth.}^{0}_{\mu}$

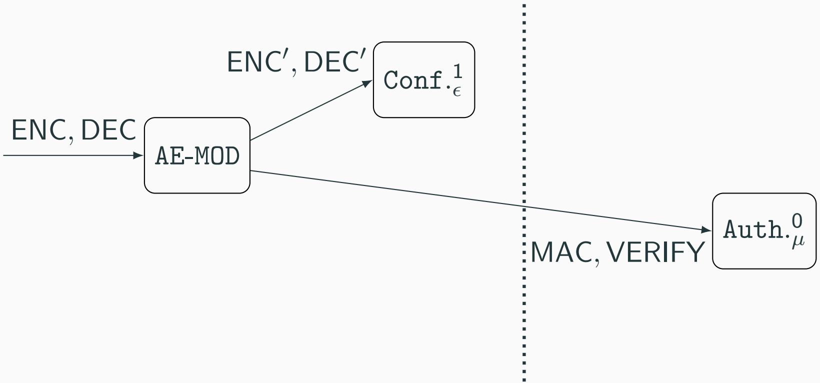
Proof


ENC, DEC
AE-MOD
$\mathrm{ENC}', \mathrm{DEC}'$
$\texttt{Conf.}_{\epsilon}^{1}$
MAC, VERIFY
$\texttt{Auth.}_{\mu}^{0}$

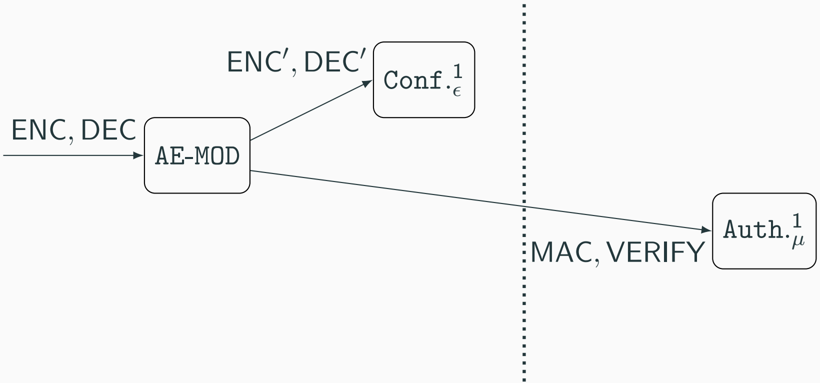
Proof



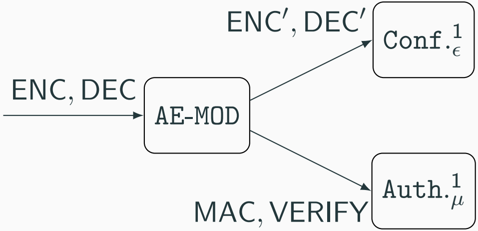
Proof



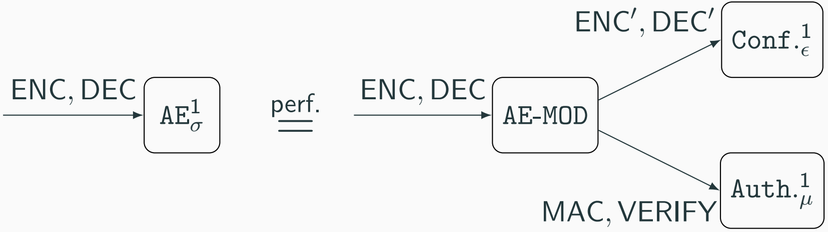
Proof



Proof



Proof con'd


ENC, DEC
$\mathtt{AE}^1_\sigma$
perf.
ENC, DEC
AE-MOD
ENC′, DEC′
$\mathtt{Conf.}^1_\epsilon$
MAC, VERIFY
$\mathtt{Auth.}^1_\mu$

Proof cont'd

$$\mathtt{AE}_\sigma^0 \overset{\text{perf.}}{=} \mathtt{AE\text{-}MOD} \to \frac{\mathtt{Conf.}_\mu^0}{\mathtt{Auth.}_\epsilon^0}$$

$$\wr\wr$$

$$\mathtt{AE}_\sigma^1 \overset{\text{perf.}}{=} \mathtt{AE\text{-}MOD} \to \frac{\mathtt{Conf.}_\mu^1}{\mathtt{Auth.}_\epsilon^1}$$

Proof cont'd

$$\mathtt{AE}_\sigma^0 \overset{\text{perf.}}{=} \mathtt{AE\text{-}MOD} \to \frac{\mathtt{Conf.}_\mu^0}{\mathtt{Auth.}_\epsilon^0}$$

$\wr\wr$ by assumptions

$$\mathtt{AE}_\sigma^1 \overset{\text{perf.}}{=} \mathtt{AE\text{-}MOD} \to \frac{\mathtt{Conf.}_\mu^1}{\mathtt{Auth.}_\epsilon^1}$$

Proof cont'd

$$\mathtt{AE}_\sigma^0 \overset{\text{perf.}}{=} \mathtt{AE\text{-}MOD} \rightarrow \frac{\mathtt{Conf.}_\mu^0}{\mathtt{Auth.}_\epsilon^0}$$

$\wr\wr$ by assumptions

$$\mathtt{AE}_\sigma^1 \overset{\text{perf.}}{=} \mathtt{AE\text{-}MOD} \rightarrow \frac{\mathtt{Conf.}_\mu^1}{\mathtt{Auth.}_\epsilon^1}$$

How to prove $\overset{\text{perf.}}{=}$?

- Inline all the code?
- Use a proof assistant?

- Functional programming language, inspired by F#, ML, OCaml and others
- Support for dependent types, refinement types and monadic effects
- Developed by Microsoft Research and Inria, as well as the F* community
- Available on Github[1]!

[1] https://github.com/FStarLang/FStar

Code Equivalence cont'd

Implement oracles in $\texttt{AE}^b$ as interface. Any code verified against that interface is code-equivalent to AE^b.

```
val enc: m:message → ST ciphertext
  (ensures
    if b=0 then
      c = sigma.enc(k,m)
    else
      c = rand (length m)
      ∧ M[c] = m
  )
```

Code Equivalence cont'd

Implement oracles in $\mathtt{AE}^b$ as interface. Any code verified against that interface is code-equivalent to AE^b.

```
val enc: m:message → ST ciphertext
  (ensures (λ s c s' ->
    if b=0 then
      c = sigma.enc(k,m)
      ∧ s == s'
    else
      c = rand (length m)
      M[c] = m
  ))
```

Code Equivalence cont'd

Implement oracles in $\mathtt{AE}^b$ as interface. Any code verified against that interface is code-equivalent to AE^b.

```
val enc: m:message → ST ciphertext
  (ensures (λ s c s' →
    if b=0 then
      c = sigma.enc(k,m)
      ∧ s == s'
    else
      c = rand (length m)
      ∧ sel s' M == upd (sel s M) c m
  ))
```

Code Equivalence cont'd

Implement oracles in $\texttt{AE}^b$ as interface. Any code verified against that interface is code-equivalent to AE^b.

```
val enc: n:nonce → m:message → ST ciphertext
  (ensures (λ s c s' →
    if b=0 then
      c = sigma.enc(k,n,m)
      ∧ s == s'
    else
      c = rand (length m)
      ∧ sel s' M == upd (sel s M) (n,c) m
  ))
```

Code Equivalence cont'd

Implement oracles in $\mathtt{AE}^b$ as interface. Any code verified against that interface is code-equivalent to AE^b.

```
val enc: n:nonce → m:message → ST ciphertext
  (requires (λ s → ∀ c . fresh M (n,c) s))
  (ensures (λ s c s' →
    if b=0 then
      c = sigma.enc(k,n,m)
      s == s'
    else
      c = rand (length m)
      ∧ sel s' M == upd (sel s M) (n,c) p
  ))
```

Code Equivalence cont'd

Implement oracles in `AE`^{b} as interface. Any code verified against that interface is code-equivalent to AE^b.

```
val enc: ap:ae_package → n:nonce → m:message
→ ST ciphertext
  (requires (λ s → ∀ c . fresh ap.M (n,c) s))
  (ensures (λ s c s' →
    if b=0 then
      c = sigma.enc(k,n,m)
      s == s'
    else
      c = rand (length m)
      ∧ sel s' ap.M == upd (sel s ap.M) (n,c) p
  ))
```

Proof cont'd

$$\mathtt{AE}_{\sigma}^{0} \overset{\text{perf.}}{=} \mathtt{AE\text{-}MOD} \rightarrow \frac{\mathtt{Conf.}_{\epsilon}^{0}}{\mathtt{Auth.}_{\mu}^{0}}$$

$$\wr\wr$$

$$\mathtt{AE}_{\sigma}^{1} \overset{\text{perf.}}{=} \mathtt{AE\text{-}MOD} \rightarrow \frac{\mathtt{Conf.}_{\epsilon}^{1}}{\mathtt{Auth.}_{\mu}^{1}}$$

Limitations

- We might be limited to a subset of protocols/security notions
- $\mathrm{F}^\star$ can't do probabilistic reasoning
- We have to change the modelling at some places to accomodate $\mathrm{F}^\star$
- Works best with real- vs ideal behaviour games

References

- Brzuska, Deligant-Lavaud, Kohbrok, Kohlweiss: State Separation for Code-Based Game-Playing Proofs
  Asiacrypt preprint.
- Bhargavan et al.: Implementing and Proving the TLS 1.3 Record Layer
  ePrint.
- A verified implementation of AEAD (slightly more complicated) here.

Future Work

- Prove the TLS 1.3 Key Schedule secure using our methodology
- Give a simple example of how to combine our methodology with $\mathrm{F}^{\star}$
- More protocols!