Universal Secret-Key and Public-Key Encryption Using Combiners

Ellie Fassman
Advised by Dr. Noah Stephens-Davidowitz

April 24, 2025

Motivation: One-Way Functions

Before discussing my own work, let's consider very simple cryptographic objects: **one-way functions.**

- A one-way function (OWF) is a function that is both easy to compute and hard to invert.
  - Easy to compute: $f(x)$ can be computed in probabilistic polynomial time (PPT)
  - Hard to invert: Any PPT adversary has at most a negligible advantage at finding the pre-image of $f(x)$

Motivation: One-Way Functions

More precisely:

- **Easy to compute.** *There exists a PPT algorithm $\mathcal{B}$ such that $\forall x \in \{0,1\}^*$, $\mathcal{B}(x) = f(x)$.*
- **Hard to invert.** *For any PPT adversary $\mathcal{A}$, there exists a negligible function $\varepsilon(n)$ such that for all $n \geq 1$,*

$$\Pr_{x \sim \{0,1\}^n} [x' \leftarrow \mathcal{A}(1^n, f(x)),\ f(x') = f(x)] \leq \varepsilon(n)\ .$$

For example, functions that rely on the computational hardness of factoring or taking a logarithm over a finite group (discrete logarithm problem).

Motivation: One-Way Functions

However, we can't prove that a function is one-way, or that one-way functions exist at all.

Why?

- In order to show a function is one-way, we would have to show that every probabilistic polynomial-time adversary has at most a negligible advantage at inverting f(x) on a random input
- In order to show the existence of OWF, we would essentially need to prove $\mathcal{P} \neq \mathcal{NP}$
- Idea: inverting a one-way function amounts to solving a computational problem that we can verify efficiently (in $\mathcal{NP}$), but can't solve efficiently (not in $\mathcal{P}$).

Motivation: Combiners for OWFs

- Say Bobby and Noah each come to me with their proposal for a OWF: f_B, f_N
- I know at least one of them is right that their function is, in fact, one-way, but I do not know who is right
- I can generate the combined function f^* where:

$$f^*(x_1, x_2) = (f_B(x_1), f_N(x_2))$$

Motivation: Combiners for OWFs

Is f^* a OWF if at least one of $\{f_B, f_N\}$ is one-way?

Motivation: Combiners for OWFs

Is f^* a OWF if at least one of $\{f_B, f_N\}$ is one-way?

No, say one of the component functions cannot be computed efficiently.

However if at least one of $\{f_B, f_N\}$ is one-way and both are efficiently computable, then f^* is a OWF. Levin generalized this idea...

First let's consider this for ℓ functions.

Combiners

Definition: A combiner takes several cryptographic primitives and produces one that inherits security from any one of them.

Example for OWFs:

$$f^*(x_1, \dots, x_\ell) = (f_1(x_1), \dots, f_\ell(x_\ell))$$

- If any f_i is one-way, then f^* is one-way.
- This is useful when we are unsure which f_i has the desired property.

Universality?

- There are many candidate OWFs (factoring, discrete logarithm, etc.)
- We assume some OWF exists, but we don't know which.
- Goal: Build a single function, f^*, that is a OWF as long as any OWF exists.

Levin's Key Idea

- Levin took this idea of a combiner to the extreme...
- Apply combiner to the **list of all Turing machines**.
- Let $T_1, T_2, \ldots$ be an enumeration of all deterministic programs.
- Define a function using outputs of first $\sqrt{n}$ machines:

$$f^*(x_1, \ldots, x_{\sqrt{n}}) = (T_1(x_1), \ldots, T_{\sqrt{n}}(x_{\sqrt{n}}))$$

- Each x_i has length $\sqrt{n}$.

If any T_i computes a hard-to-invert function, it will eventually appear in this list.

Levin's Key Idea: Universality

The Big Idea

Use a combiner not over known functions, but over the **infinite set of all efficient algorithms**.

- Enumerate all Turing machines??!!
- If a OWF exists, it will be found, and the resulting function f^* will also be one-way

The Halting Problem Issue

- Some T_i may not halt or run in non-polynomial time.
- Makes f^* undefined or inefficient.
- Need a workaround to make f^* well-defined and efficient.

Solution: Time-bounded computation

$$T_i^{t(n)}(x) = \begin{cases} T_i(x) & \text{if } T_i \text{ halts in } t(|x|) \text{ steps} \\ 0 & \text{otherwise} \end{cases}$$

Levin's Universal OWF: Definition

- Use bounded versions of machines:

$$f_U(x_1, \ldots, x_{\sqrt{n}}) = \left(T_1^{n^5}(x_1), \ldots, T_{\sqrt{n}}^{n^5}(x_{\sqrt{n}}) \right)$$

- $T_i^{n^5}$ halts within n^5 steps or returns 0
- Function is computable in time $O(n^6)$

Crucial idea: Every efficient OWF will eventually be simulated by some T_i in this list.

Note that the choices of $\sqrt{n}$ and n^5 can be generalized to any function $p(n) = n^C$ for sufficiently large C.

Sketch of Proof

Statement: If there exists an OWF f computable in $O(n^4)$ time, then f_U is also one-way.

Proof Intuition:

- For large enough n, $n^5 > O(n^4)$
- Suppose T_{i^*} computes f and runs in $O(n^4)$.
- For large n, T_{i^*} appears among first $\sqrt{n}$ machines.
- We can prove that f_U is one-way via the contrapositive: If there exists an adversary breaks f_U, we can construct an adversary that breaks f.

Wrapping Up OWFs

- We can remove this time bound which shows that any OWF implies a OWF running in $O(n^4)$ (will show later).
- f_U is one-way if *any* efficiently computable OWF exists
- Based on enumerating and simulating all efficient functions.

From OWFs to Encryption

- Levin's approach motivates our work on encryption
- Replace "functions" with "encryption schemes"
- Instead of inverting, adversaries try to distinguish messages
- Goal: construct encryption that is secure if any scheme in the candidate list is.

Our Contribution

- Efficient combiner constructions for SKE and PKE.
- Output is correct and semantically secure IFF any input is.
- Universality: if a secure encryption scheme exists, our construction is secure.

Secret-Key Encryption (SKE)

A secret-key encryption scheme consists of three efficiently computable algorithms:

- $Gen(1^n) \rightarrow sk$: Generate a secret key
- $Enc(sk, m) \rightarrow c$: Encrypt a message m
- $Dec(sk, c) \rightarrow m$: Decrypt the ciphertext c

Correctness: $Dec(sk, Enc(sk, m)) = m$

Security for SKE

Definition of Many Message Semantic Security in the SKE Context

Adversary ($\mathcal{A}$)		**Challenger ($\mathcal{C}$)**
	$\xrightarrow{M_0 \in \{0,1\}^K, M_1 \in \{0,1\}^K}$	
		$sk \leftarrow \mathsf{Gen}(1^n)$
		$b \sim \{0,1\}$
		$c_i \leftarrow \mathsf{Enc}(sk, M_{b,[i]})$
	$\xleftarrow{c_1, \ldots, c_K}$	
	$\xrightarrow{b'}$	
		If $b = b'$, output WIN.

Why is this useful?

Combiner for SKE: Naive Approach

- Say Bobby and Noah each come to me with their proposal for a secure SKE scheme: $(Gen_B, Enc_B, Dec_B), (Gen_N, Enc_N, Dec_N)$
- I know at least one of them is right that their scheme is secure, but I do not know who is right
- I can create a combined scheme...

Combiner for SKE: Naive Approach

$$Gen^*(1^n) := (Gen_B(1^n), Gen_N(1^n))$$

$$Enc^*((sk_B, sk_N), (m_1, m_2)) := (Enc_B(sk_B, m_1), Enc_N(sk_N, m_2))$$

$$Dec^*((sk_B, sk_N), (c_1, c_2)) := (Dec_B(sk_B, c_1), Dec_N(sk_N, c_2))$$

Combiner for SKE: Naive Approach

- (Gen^*, Enc^*, Dec^*) is not necessarily secure, even if at least one of the two component schemes is.
- If one of the component schemes sent its plaintext in the clear, this would give an adversary a non-negligible advantage at guessing the original plaintext.
- In fact, not just a non-negligible advantage, but rather an adversary could always guess the message list that was encrypted (with probability 1).

Secret-Key Combiner: Overview

- Let's generalize from two schemes to ℓ schemes.
- Encrypt random bits $b_1, \ldots, b_\ell$ with each scheme
- XOR these unencrypted bits with the message bit $m \in \{0, 1\}$ to produce tag d
- Decryption XORs decrypted bits with tag d to recover m
- These ℓ bits serve as one-time pads for each of the ℓ schemes.

Secret-Key Combiner: Key Generation

- Gen* generates ℓ keys: $sk_i \leftarrow Gen_i(1^n)$
- Output: $sk^* = (sk_1, \ldots, sk_\ell)$

Secret-Key Combiner: Encryption

- Flip ℓ random bits $b_1, \ldots, b_\ell$
- Encrypt b_i with $Enc_i(sk_i, b_i)$
- Compute tag $d = b_1 \oplus \ldots \oplus b_\ell \oplus m$
- Output: $c^* = (c_1, \ldots, c_\ell, d)$

Secret-Key Combiner: Decryption

- Decrypt $b_i = Dec_i(sk_i, c_i)$ for all i
- Output: $b_1 \oplus \ldots \oplus b_\ell \oplus d$

All together

- $\mathsf{Gen}^*(1^n) :=$

  Output $sk^* := (sk_1, \ldots, sk_\ell)$, where $sk_i \leftarrow \mathsf{Gen}'_i(1^n)$.

- $\mathsf{Enc}^*(sk^*, m \in \{0,1\}) :=$

  Flip ℓ coins $b_1, \ldots, b_\ell$

  Then for each b_i compute $c_i \leftarrow \mathsf{Enc}'_i(sk_i, b_i)$

  Let d be the XOR of these coins with the plaintext, that is, $d := b_1 \oplus \ldots \oplus b_\ell \oplus m$

  Output $c^* := (c_1, \ldots, c_\ell, d)$.

- $\mathsf{Dec}^*(sk^*, c^*) :=$

  For each $i \in [\ell]$, compute $b_i \leftarrow \mathsf{Dec}'_i(sk_i, c_i)$

  Output $b_1 \oplus \ldots \oplus b_\ell \oplus d$ which will result in the plaintext bit m.

Issue: If one of the component schemes is incorrect (does not decrypt correctly), then the combined scheme is incorrect. This is where the prime schemes come in!

Ensuring Correctness for SKE Combiner

In order to ensure correctness of the combiner, we must define (Gen'_i, Enc'_i, Dec'_i) for each scheme i.

- $\mathsf{Gen}'_i(1^n) := \mathsf{Gen}_i(1^n)$.
- $\mathsf{Enc}'_i(sk_i, m_i) :=$

  Compute $c_i \leftarrow \mathsf{Enc}_i(sk_i, m_i)$

  Check if $\mathsf{Dec}_i(sk_i, c_i) = m_i$

  If so, output $c'_i := (1, c_i)$

  Otherwise, output $c'_i := (0, m_i)$.
- $\mathsf{Dec}'_i(sk_i, (\alpha, c_i)) :=$

  Check if bit $\alpha = 1$

  If so, output $\mathsf{Dec}_i(sk_i, c_i)$

  Otherwise, output c_i.

Security Theorem for SKE

Now we can show Theorem 2.1:

Theorem 2.1. $(\mathsf{Gen}^*, \mathsf{Enc}^*, \mathsf{Dec}^*)$ *is many message semantically secure and correct, assuming that at least one of the* ℓ *schemes is many message semantically secure and correct.*

Note that we must show that the combined scheme is **correct, secure,** and **efficiently computable**.

- We have already shown correctness.
- We will now prove security.
- We will handle the efficiency later.

Security of SKE Combiner

Proof Idea:

- Use adversary A against the combiner to build A' against some secure scheme i
- Contrapositive: If combiner is insecure (A has an advantage at breaking the combined scheme), then A' breaks i

Universal SKE Construction

- Enumerate all SKE schemes that run in poly(n)
- Use timeout (e.g., $O(n^2)$) to reject inefficient ones
- Apply combiner to the first n elements in this list

Handling Runtime

- If a secure scheme runs in $O(n^C)$, scale the security parameter (1^n) to make it $O(n^2)$
- Use padding to ensure security
- Ensures resulting universal scheme is efficient

Universal SKE Construction

- We are enumerating through all triples of Turing machines!!!
- If a secure SKE scheme exists, for large enough n it will appear in the list and be inputted into the combiner!

Public-Key Encryption (PKE)

A public-key encryption scheme consists of three efficiently computable algorithms:

- $Gen(1^n) \rightarrow (pk, sk)$: Generate a public key-secret key pair
- $Enc(pk, m) \rightarrow c$: Encrypt a message m
- $Dec(sk, c) \rightarrow m$: Decrypt the ciphertext c

Correctness: $Dec(sk, Enc(pk, m)) = m$ with high probability

PKE is Amazing

- Each person has a **public key** and a **secret key**.
- Public key is shared with the world.
- Messages encrypted with the public key can *only* be decrypted with the secret key.

No shared secrets needed in advance.

Security for PKE

Definition of Semantic Security in the PKE Context

Adversary ($\mathcal{A}$)		**Challenger** ($\mathcal{C}$)
		$(pk, sk) \leftarrow \mathsf{Gen}(1^n)$
	$\xleftarrow{\quad pk \quad}$	
	$\xrightarrow{\quad m_0, m_1 \quad}$	
		$b \sim \{0, 1\}$
		$c \leftarrow \mathsf{Enc}(pk, m_b)$
	$\xleftarrow{\quad c \quad}$	
	$\xrightarrow{\quad b' \quad}$	
		If $b = b'$, output WIN.

Why is this useful?

PKE Combiner: Construction

- $\mathsf{Gen}^*(1^n) :=$

  For $i \in [\ell]$:

  Compute $(pk_i, sk_i) \leftarrow \mathsf{Gen}'_i(1^n)$

  Output $(pk^*, sk^*) := ((pk_1, sk_1), \ldots, (pk_\ell, sk_\ell))$.

- $\mathsf{Enc}^*(pk^*, m \in \{0,1\}) :=$

  Flip ℓ coins $b_1, \ldots, b_\ell$

  Then for each b_i, compute $c_i \leftarrow \mathsf{Enc}'_i(pk_i, b_i)$

  Compute $d := b_1 \oplus \ldots \oplus b_\ell \oplus m$

  Output $c^* := (c_1, \ldots, c_\ell, d)$.

- $\mathsf{Dec}^*(sk^*, c^* := (c_1, \ldots, c_\ell, d)) :=$

  For each $i \in [\ell]$, compute $b_i \leftarrow \mathsf{Dec}'_i(sk_i, c_i)$

  Output $b_1 \oplus \ldots \oplus b_\ell \oplus d$, which will result in the plaintext bit m.

Issue: If one of the component schemes is incorrect (does not decrypt correctly), then the combined scheme is incorrect. This is where the prime schemes come in!

Ensuring Correctness for PKE Combiner

In order to ensure correctness of the combiner, we must define (Gen'_i, Enc'_i, Dec'_i) for each scheme i.

- $\mathsf{Gen}'_i(1^n) :=$

  Compute $(pk_i, sk_i) \leftarrow \mathsf{Gen}_i(1^n)$

  For $j \in [n]$:

  Sample $b_j \sim \{0,1\}$

  Compute $c_j \leftarrow \mathsf{Enc}_i(pk_i, b_j), b'_j \leftarrow \mathsf{Dec}_i(sk_i, c_j)$

  If $b_j \neq b'_j$, $f \leftarrow 1$

  If $f = 1$, output $(sk'_i, pk'_i) := (0, 0)$

  Otherwise, output $(sk'_i, pk'_i) := ((1, pk_i), (1, sk_i))$.

- $\mathsf{Enc}'_i(pk'_i, m_i) :=$

  If $pk_i = 0$, output $c' := m_i$

  Otherwise for $j \in [n]$:

  Compute $c_j \leftarrow \mathsf{Enc}_i(pk_i, m_i)$

  Output $c' := (c_1, \ldots, c_n)$.

- $\mathsf{Dec}'_i(sk'_i, c' := (c_1, \ldots c_n)) :=$

  If $sk'_i = 0$, output c_1

  Otherwise for $j \in [n]$:

  Compute $b_j \leftarrow \mathsf{Dec}_i(sk'_i, c_j)$

  Output $\mathsf{maj}(b_1, \ldots, b_n)$.

Ensuring Correctness for PKE Combiner

- For each scheme i, Gen'_i runs tests on each key pair (pk_i, sk_i)
- If decryption fails with high probability, output a placeholder key
- Only keep validated key pairs
- Decryption: majority vote on decrypted bits

Security Theorem for PKE

Now we can show Theorem 5.2:

Theorem 5.2. $(\mathsf{Gen}^*, \mathsf{Enc}^*, \mathsf{Dec}^*)$ *is semantically secure and correct, assuming that at least one of the* ℓ *schemes is semantically secure and correct.*

Note that we must show that the combined scheme is **correct, secure,** and **efficiently computable**.

- We ensure correctness with the definition of the prime schemes.
- We will now prove security.
- We will handle the efficiency later.

Security of PKE Combiner

Proof Idea:

- Use adversary A against the combiner to build A' against some secure scheme i
- Contrapositive: If combiner is insecure (A has an advantage at breaking the combined scheme), then A' breaks i

Universal PKE Construction

- Enumerate all PKE schemes that run in poly(n)
- Use timeout (e.g., $O(n^2)$) to reject inefficient ones
- Apply combiner to the the first n elements in this list
- The way we handled runtime for SKE works the same for PKE

Universal PKE Construction

- We are enumerating through all triples of Turing machines!!!
- If a secure PKE scheme exists, for large enough n it will appear in the list and be inputted into the combiner!

Final Theorems

SKE: If there exists any poly-time, correct and secure SKE scheme, then the combined SKE scheme is also correct, secure, and poly-time.

PKE: Same holds for public-key encryption: universal PKE exists if *any* secure PKE exists.

No assumptions needed beyond the existence of one secure scheme.

Thank You!

Questions?

oef3@cornell.edu