No Way, JOSE!

Designing Cryptography Features for Mere Mortals
Scott Arciszewski

- **Paragon Initiative Enterprises, LLC**
  - Software development (open source)
    - The person to blame for getting libsodium into PHP 7.2
    - Also wrote the sodium_compatible polyfill for PHP 5.2 – 7.1
    - Many PHP security libraries
  - Security research
    - Handfuls of CVEs
    - Sometimes published on Full Disclosure

- **Twitter handle:** @CiPHPerCoder
Why are we here today?
Why are we here today?

Why You Don't Roll Your Own Crypto

The golden rule of encryption.
Why are we here today?

Google search results for "don't roll your own crypto".
Why are we here today?

- Everyone knows “Don’t roll your own crypto”
Why are we here today?

- Everyone knows “Don’t roll your own crypto”
  - Amateurs produce amateur cryptography
Why are we here today?

- Everyone knows “Don’t roll your own crypto”
  - Amateurs produce amateur cryptography
  - It’s extremely difficult to get right
Why are we here today?

• Everyone knows “Don’t roll your own crypto”
  − Amateurs produce amateur cryptography
  − It’s extremely difficult to get right
  − Even experts make mistakes
Why are we here today?

• Everyone knows “Don’t roll your own crypto”
  – Amateurs produce amateur cryptography
  – It’s extremely difficult to get right
  – Even experts make mistakes
  – Cryptography should be a collaborative practice in which many experts vet each others’ designs
Why are we here today?

• Everyone knows “Don’t roll your own crypto”
  – Amateurs produce amateur cryptography
  – It’s extremely difficult to get right
  – Even experts make mistakes
  – Cryptography should be a collaborative practice in which many experts vet each others’ designs

• The problem: the buck usually stops there.
Why are we here today?

- “What should I do instead of rolling my own?”
Why are we here today?

• “What should I do instead of rolling my own?”
  – Bad outcome: “Use RSAES-OAEP with SHA256 and MGF1+SHA256 bzzrt pop ffsssssssst exponent 65537” (h/t Latacora)
Why are we here today?

• “What should I do instead of rolling my own?”
  – Bad outcome: “Use RSAES-OAEP with SHA256 and MGF1+SHA256 bzzrt pop ffsssssst exponent 65537” (h/t Latacora)
• Developers need cryptography features to solve problems.
Why are we here today?

• “What should I do instead of rolling my own?”
  – Bad outcome: “Use RSAES-OAEP with SHA256 and MGF1+SHA256 bzzrt pop ffsssssssst exponent 65537” (h/t Latacora)

• Developers need cryptography features to solve problems.

• If we don’t want them rolling their own, they need easy-to-use tools that don’t open the door to a ton of attacks.
Case Study: JOSE

- Javascript Object Signing and Encryption
Case Study: JOSE

- Javascript Object Signing and Encryption
- A family of standards (with IETF RFCs) that define JSON Web Tokens (JWT), JSON Web Signatures (JWS), JSON Web Encryption (JWE), etc.
Case Study: JOSE

- Javascript Object Signing and Encryption
- A family of standards (with IETF RFCs) that define JSON Web Tokens (JWT), JSON Web Signatures (JWS), JSON Web Encryption (JWE), etc.
- Most developers that use JOSE focus on JWT.
Case Study: JOSE

- Javascript Object Signing and Encryption
- A family of standards (with IETF RFCs) that define JSON Web Tokens (JWT), JSON Web Signatures (JWS), JSON Web Encryption (JWE), etc.
- Most developers that use JOSE focus on JWT.
- Uses:
Case Study: JOSE

- **Javascript Object Signing and Encryption**
- A family of standards (with IETF RFCs) that define JSON Web Tokens (JWT), JSON Web Signatures (JWS), JSON Web Encryption (JWE), etc.
- Most developers that use JOSE focus on JWT.
- Uses:
  - Short-lived claims (usually signed by a third party)
Case Study: JOSE

- **Javascript Object Signing and Encryption**
- A family of standards (with IETF RFCs) that define JSON Web Tokens (JWT), JSON Web Signatures (JWS), JSON Web Encryption (JWE), etc.
- Most developers that use JOSE focus on JWT.
- Uses:
  - Short-lived claims (usually signed by a third party)
  - A laundry list of misuse
JSON Web Tokens

• Quoth the RFC:
  - JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.
JSON Web Tokens

- Quoth the RFC:
  - JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.

- Translation: a JWT uses JWE or JWS.
JSON Web Tokens

- Quoth the RFC:
  - JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.

- Translation: a JWT uses JWE or JWS.
  - Consequently, JWS/JWE security flaws are almost always relevant to JWT.
JSON Web Token (structure)

Above: [https://jwt.io](https://jwt.io) (a tool from Auth0)
JSON Web Signatures

- The "alg" header
  - Defines what algorithm the token uses
JSON Web Signatures

- The “alg” header
  - Defines what algorithm the token uses
    - HS256 = HMAC-SHA256
JSON Web Signatures

• The “alg” header
  - Defines what algorithm the token uses
    • HS256 = HMAC-SHA256
    • RS256 = RSA with PKCS1v1.5 and SHA256
JSON Web Signatures

- The “alg” header
  - Defines what algorithm the token uses
    - HS256 = HMAC-SHA256
    - RS256 = RSA with PKCS1v1.5 and SHA256
    - none = \_(ツ)_/\
JSON Web Signatures

• The “alg” header
  - Defines what algorithm the token uses
    • HS256 = HMAC-SHA256
    • RS256 = RSA with PKCS1v1.5 and SHA256
    • none = ¯\_(ツ)_/¯
  - Mixes symmetric with asymmetric cryptography
JSON Web Signatures

• The “alg” header
  - Defines what algorithm the token uses
  - HS256 = HMAC-SHA256
  - RS256 = RSA with PKCS1v1.5 and SHA256
  - none = `\_(ツ)_/` – Mixes symmetric with asymmetric cryptography
  - Attackers can alter tokens and choose this header
JSON Web Signatures

• It gets worse.
It gets worse.

RFC 7515 section 4.1.1:
- “This Header Parameter MUST be present and MUST be understood and processed by implementations.”
JSON Web Signatures

- It gets worse.
- RFC 7515 section 4.1.1:
  - “This Header Parameter MUST be present and MUST be understood and processed by implementations.”
  - To a developer, “understood and processed” means “obeyed”.
It gets worse.

RFC 7515 section 4.1.1:

- “This Header Parameter MUST be present and MUST be understood and processed by implementations.”
- To a developer, “understood and processed” means “obeyed”.

This has led to critical vulnerabilities in JWT libraries. (CVE-2015-2964, etc.)
JSON Web Encryption

• Key encryption options:
  - RSA with PKCS #1 v1.5 padding
  - RSA with OAEP padding
  - ECDH-ES
  - AES-GCM
JSON Web Encryption

• Key encryption options:
  - RSA with PKCS #1 v1.5 padding *(asym)*
  - RSA with OAEP padding *(asym)*
  - ECDH-ES *(asym)*
  - AES-GCM *(sym)*
JSON Web Encryption

- Key encryption options:
  - RSA with PKCS #1 v1.5 padding (asym)
  - RSA with OAEP padding (asym)
  - ECDH-ES (asym)
  - AES-GCM (sym)

- One of these things is not like the other.
JSON Web Encryption

• The way ECDH-ES is specified opens the door to invalid curve attacks, which allows attackers to recover your private key remotely.
The way ECDH-ES is specified opens the door to invalid curve attacks, which allows attackers to recover your private key remotely.

What JOSE does:
- Expects x and y coordinates in the token (which is provided by attackers)
JSON Web Encryption

- The way ECDH-ES is specified opens the door to invalid curve attacks, which allows attackers to recover your private key remotely.

- What JOSE does:
  - Expects x and y coordinates in the token (which is provided by attackers)

- What JOSE should have done:
JSON Web Encryption

• The way ECDH-ES is specified opens the door to invalid curve attacks, which allows attackers to recover your private key remotely.

• What JOSE does:
  – Expects x and y coordinates in the token (which is provided by attackers)

• What JOSE should have done:
  – Expect an x coordinate and a single bit for the sign of y.
The way ECDH-ES is specified opens the door to invalid curve attacks, which allows attackers to recover your private key remotely.

What JOSE does:
- Expects x and y coordinates in the token (which is provided by attackers)

What JOSE should have done:
- Expect an x coordinate and a single bit for the sign of y.
- Failing that, making point validation explicit.
The Generalized Problem

• “Let’s give developers options!”
The Generalized Problem

• “Let’s give developers options!”
  – This leads to a condition called Reasoning By Lego.
The Generalized Problem

- “Let’s give developers options!”
  - This leads to a condition called Reasoning By Lego.
  - Anybody remember “MAC and Encrypt” cipher constructions?
The Generalized Problem

- "Let’s give developers options!"
  - This leads to a condition called Reasoning By Lego.
    - Anybody remember “MAC and Encrypt” cipher constructions?
- Imagine you’re tasked with a brick wall.
The Generalized Problem

• “Let’s give developers options!”
  – This leads to a condition called Reasoning By Lego.
    • Anybody remember “MAC and Encrypt” cipher constructions?

• Imagine you’re tasked with a brick wall.
  – Twist: There’s a mesh of mortar laid out for you, and you have to slide bricks into place.
The Generalized Problem

• “Let’s give developers options!”
  – This leads to a condition called Reasoning By Lego.
    • Anybody remember “MAC and Encrypt” cipher constructions?
• Imagine you’re tasked with a brick wall.
  – Twist: There’s a mesh of mortar laid out for you, and you have to slide bricks into place.
  – The architects insist this lets you freely swap out clay bricks with concrete bricks if termites adapt to eat clay, or vice versa.
The Generalized Problem

- “Let’s give developers options!”
  - This leads to a condition called Reasoning By Lego.
    - Anybody remember “MAC and Encrypt” cipher constructions?
  - Imagine you’re tasked with a brick wall.
    - Twist: There’s a mesh of mortar laid out for you, and you have to slide bricks into place.
    - The architects insist this lets you freely swap out clay bricks with concrete bricks if termites adapt to eat clay, or vice versa.
    - Would you trust that wall to hold up the roof?
“To pwn, or JWT to pwn?”

- There are a lot of ways for JWTs to go wrong baked into the JOSE standards
“To pwn, or JWT to pwn?”

• There are a lot of ways for JWTs to go wrong baked into the JOSE standards
  – I’m not even getting into implementation-specific security risks or user error.
“To pwn, or JWT to pwn?”

- There are a lot of ways for JWTs to go wrong baked into the JOSE standards
  - I’m not even getting into implementation-specific security risks or user error.

- The JOSE advocate response to this criticism is “use [a specific library]”
“To pwn, or JWT to pwn?”

- There are a lot of ways for JWTs to go wrong baked into the JOSE standards
  - I’m not even getting into implementation-specific security risks or user error.

- The JOSE advocate response to this criticism is “use [a specific library]”
  - This shifts the blame onto the library developers and the library’s users (i.e. developers)
“To pwn, or JWT to pwn?”

- There are a lot of ways for JWTs to go wrong baked into the JOSE standards
  - I’m not even getting into implementation-specific security risks or user error.
- The JOSE advocate response to this criticism is “use [a specific library]”
  - This shifts the blame onto the library developers and the library’s users (i.e. developers)
- If we want secure systems, this is an antipattern!
Industry Antipatterns

- Standard designers:
  - Let’s give users a lot of choices.
Industry Antipatterns

- Standard designers:
  - Let’s give users a lot of choices.

- Advocates:
  - Blame the implementation, rather than the standard!
Industry Antipatterns

- **Standard designers:**
  - Let’s give users a lot of choices.

- **Advocates:**
  - Blame the implementation, rather than the standard!

- **Security experts:**
  - Declare a standard harmful, provide no alternative
Industry Antipatterns

- Standard designers:
  - Let’s give users a lot of choices.
- Advocates:
  - Blame the implementation, rather than the standard!
- Security experts:
  - Declare a standard harmful, provide no alternative
- Developers:
  - Roll their own crypto
Industry Antipatterns

- **Standard designers:**
  - Let’s give users a lot of choices.

- **Advocates:**
  - Blame the implementation, rather than the standard!

- **Security experts:**
  - Declare a standard harmful, provide no alternative

- **Developers:**
  - Roll their own crypto
  - ...can you *really* blame them?
Goal: Stop developers from rolling their own cryptography
Goal: Stop developers from rolling their own cryptography

My proposal: Design a better standard that is a lot easier to use securely than to use insecurely
PASETO

- Platform-Agnostic SEcurity TOkens
PASETO

- Platform-Agnostic SEcurity TOkens
  - Pronounce: Paw Set Oh
  - $\{ x \in IR \} \quad O(n)$
PASETO

- **Platform-Agnostic SEcurtiy TOkens**
- Pronounce: Paw Set Oh
- $\{ x \in \mathbb{R} \}$ \( O(n) \)
- Design goals
PASETO

- **Platform-Agnostic SEcurity TOkens**
  - Pronounce: Paw Set Oh
  - $\{ x \in \mathbb{R} \}$ $O(n)$

- **Design goals**
  - Minimize runtime negotiation
PASETO

- **Platform-Agnostic SECurtiy Tokens**
  - Pronounce: Paw Set Oh
  - $\{ x \in \mathbb{R} \} \quad O(n)$

- Design goals
  - Minimize runtime negotiation
  - Versioned tokens (forward-compatible)
PASETO

- **Platform-Agnostic SEcurtiy TOkens**
  - Pronounce: Paw Set Oh
  - $\{ x \in \mathbb{R} \} \quad O(n)$

- **Design goals**
  - Minimize runtime negotiation
  - Versioned tokens (forward-compatible)
  - “One True Ciphersuite” for each version
PASETO

- Platform-Agnostic SEcurity TOkens
  - Pronounce: Paw Set Oh
  - \( \{ x \in \mathbb{IR} \} \quad O(n) \)
- Design goals
  - Minimize runtime negotiation
  - Versioned tokens (forward-compatible)
  - “One True Ciphersuite” for each version
  - Less knobs and levers for end users
PASETO Overview

- Token Structure
PASETO Overview

- Token Structure
  - Version (v1, v2)
PASETO Overview

• Token Structure
  − Version (v1, v2)
  − Purpose (local, public)
PASETO Overview

- Token Structure
  - Version (v1, v2)
  - Purpose (local, public)
  - Payload
PASETO Overview

- Token Structure
  - Version (v1, v2)
  - Purpose (local, public)
  - Payload
  - Footer (optional)
PASETO Overview

- **Token Structure**
  - Version (v1, v2)
  - Purpose (local, public)
  - Payload
  - Footer (optional)

- **Payload and optional footer are Base64url encoded (as specified in RFC 4648)**
PASETO Overview

• Version
  - Defines the ciphersuite for each distinct purpose
PASETO Overview

- Version
  - Defines the ciphersuite for each distinct purpose
- Purpose
PASETO Overview

• Version
  - Defines the ciphersuite for each distinct purpose
• Purpose
  - Local: Symmetric-key authenticated encryption
PASETO Overview

- Version
  - Defines the ciphersuite for each distinct purpose
- Purpose
  - Local: Symmetric-key authenticated encryption
  - Public: Asymmetric-key digital signatures
PASETO Overview

- **Version**
  - Defines the ciphersuite for each distinct purpose

- **Purpose**
  - Local: Symmetric-key authenticated encryption
  - Public: Asymmetric-key digital signatures

- **Footer (optional)**
PASETO Overview

- **Version**
  - Defines the ciphersuite for each distinct purpose

- **Purpose**
  - Local: Symmetric-key authenticated encryption
  - Public: Asymmetric-key digital signatures

- **Footer (optional)**
  - Authenticated. Useful for key rotation schemes.
PASETO Overview

- Version
  - Defines the ciphersuite for each distinct purpose
- Purpose
  - Local: Symmetric-key authenticated encryption
  - Public: Asymmetric-key digital signatures
- Footer (optional)
  - Authenticated. Useful for key rotation schemes.
- Envisioned use cases:
PASETO Overview

• Version
  – Defines the ciphersuite for each distinct purpose

• Purpose
  – Local: Symmetric-key authenticated encryption
  – Public: Asymmetric-key digital signatures

• Footer (optional)
  – Authenticated. Useful for key rotation schemes.

• Envisioned use cases:
  – Short-lived, one-time third-party access tokens
PASETO Overview

• Version
  - Defines the ciphersuite for each distinct purpose

• Purpose
  - Local: Symmetric-key authenticated encryption
  - Public: Asymmetric-key digital signatures

• Footer (optional)
  - Authenticated. Useful for key rotation schemes.

• Envisioned use cases:
  - Short-lived, one-time third-party access tokens
  - OpenID Connect
PASETTO Versions

• Version 1: Compatibility mode
  − Meant to work when only OpenSSL is available
PASETO Versions

- Version 1: Compatibility mode
  - Meant to work when only OpenSSL is available
  - Local: AES-256-CTR + HMAC-SHA384 (EtM)
    Keys split with HKDF-HMAC-SHA384
PASETO Versions

- Version 1: Compatibility mode
  - Meant to work when only OpenSSL is available
    - Local: AES-256-CTR + HMAC-SHA384 (EtM)
      Keys split with HKDF-HMAC-SHA384
    - Public: RSASSA-PSS, $e=65537$, SHA384 and MGF1+SHA384, with 2048-bit keys
PASETO Versions

• Version 1: Compatibility mode
  – Meant to work when only OpenSSL is available
    • Local: AES-256-CTR + HMAC-SHA384 (EtM)
      Keys split with HKDF-HMAC-SHA384
    • Public: RSASSA-PSS, e=65537, SHA384 and MGF1+SHA384, with 2048-bit keys

• Version 2: Recommended
PASETO Versions

• Version 1: Compatibility mode
  - Meant to work when only OpenSSL is available
    • Local: AES-256-CTR + HMAC-SHA384 (EtM)
      Keys split with HKDF-HMAC-SHA384
    • Public: RSASSA-PSS, e=65537, SHA384 and MGF1+SHA384, with 2048-bit keys

• Version 2: Recommended
  - Uses libsodium (or a compatible implementation)
    • Local: XChaCha20-Poly1305
    • Public: Ed25519
PASETO Overview

• Example:
  - Payload: “foo”
  Footer: “bar”
PASETO Overview

• Example:
  - Payload: “foo”
  Footer: “bar”
  - v2.local.XRweHw55LcYDJ_pFGo2zWlhX-gGpTTIowCuSHQ88N2MvUqoNZJNYex7A.YmFy
PASETO Overview

● Example:
  - Payload: “foo”
  Footer: “bar”
  - v2.local.xRweHw55LcYDJ_pFGo2zWlhX-gGpTTIAowCuSHQ88N2MvUpqoNZJNYex7A.YmFy
    - k = 0xa71913ea1750aa39142e00089dccc7990da5173521b6201c4badd460b1f50ab0
PASETO Overview

- Example:
  - Payload: “foo”
  - Footer: “bar”
  - v2.local.xRweHw55LcYDJ_pFGo2zWIhX-gGpTTIAowCuSHQ88N2MvUpqoNZJNYex7A.YmFy
    - k = 0xa71913ea1750aa39142e00089dce47990da5173521b6201c4badd460b1f50ab0
  - v2.public.Zm9vknDoCUzU05m6yyiYFFQcsO9WnBJPjatGpfL20kyb9Q_abkUcSa-Pwzmn8fCuc6kYpmAkOz3e9WzMgyqhMb1CA.YmFy
PASETO Overview

- **Example:**
  - Payload: “foo”
  - Footer: “bar”
  - v2.local.xRweHw55LcYDJ_pFGo2zWIhX-gGpTTIAowCuSHQ88N2MvUpqoNZJNYex7A.YmFy
    - k = 0xa71913ea1750aa39142e00089dCc47990da5173521b6201c4badd460b1f50ab0
  - v2.public.Zm9vknDoCUzU05m6yyiYFFQcsO9WnBJPjatGpfL2Okyb9Q_abkUcSa-Pwzmn8fCuc6kYpmAkOz3e9WzMgyqhMb1CA.YmFy
    - pk = 0x72bbbing1c8b77b1e5d71e7ec11f3b53cc69097757053b530a035237c2e278a33d
PASETO Overview

• Example:
  - Payload: “foo”
  - Footer: “bar”
    - v2.local.xRweHw55LcYDJ_pFGo2zWIhX-gGpTTIAowCuSHQ88N2MvUpqoNZJNYex7A.YmFy
      • k = 0xa71913ea1750aa39142e00089dcc47990da5173521b6201c4badd460b1f50ab0
    - v2.public.Zm9vknDoCUzU05m6yyiYFFQcsO9WnBJPjatGpfL2Okyb9Q_abkUcSa-Pwzmn8fCuc6kYpmAkOz3e9WzMgyqhMb1CA.YmFy
      • pk = 0x72bbbb1c8b77b1e5d71e7ec11f3b53cc69097757053b530a035237c2e278a33d
      • sk = 0x65383a773dd019c00a83c4f113acc8b1b2c114a10be230bae9fc935164ab34472bbbb1c8b77b1e5d71e7ec11f3b53cc69097757053b530a035237c2e278a33d
PASETO Overview

• Example without a footer:
  - Payload: “foo”
  Footer: NULL
  - v2.local.0mdhIsOmc4H5kWCBX5Tdty1jX-tzyvJclRptsvhqtdQD9P9gb1OPsSXb8Q
    • k = 0xa71913ea1750aa39142e00089dcc47990da5173521b6201c4badd460b1f50ab0
  - v2.public.Zm9vybtfJiXsVkfXsW8JW_Fb-mpAspqVZ9cpTtmvHdYrDaWnIZp1cf0jFB9NXe-SujwmwXpvVl0pJM0GSCTzOguAA
    • pk = 0x72bbbb1c8b77b1e5d71e7ec11f3b53cc690977757053b530a035237c2e278a33d
    sk = 0x65383a773dd0191c00a83c4f113acc8b1b2c114a10bc230bae9fc935164ab34472bbbb1c8b77b1e5d71e7ec11f3b53cc690977757053b530a035237c2e278a33d
PASETO Internals

End-users don’t need to know this stuff
PASETO – Rules for All Versions

- Formal specification regarding key/nonce-gen:
PASETO – Rules for All Versions

• Formal specification regarding key/nonce-gen:
  – Just use urandom. Userspace PRNGs are forbidden. No more Mersenne Twister or LCGs.
PASETTO – Rules for All Versions

- Formal specification regarding key/nonce-gen:
  - Just use urandom. Userspace PRNGs are forbidden. No more Mersenne Twister or LCGs.
- No IND-CCA2 insecure public key cryptography
PASETO – Rules for All Versions

- Formal specification regarding key/nonce-gen:
  - Just use `urandom`. Userspace PRNGs are forbidden. No more Mersenne Twister or LCGs.

- No IND-CCA2 insecure public key cryptography
  - PKCS #1 v1.5 is explicitly forbidden, forever
PASETO – Rules for All Versions

• Formal specification regarding key/nonce-gen:
  – Just use urandom. Userspace PRNGs are forbidden. No more Mersenne Twister or LCGs.

• No IND-CCA2 insecure public key cryptography
  – PKCS #1 v1.5 is explicitly forbidden, forever
    • Bleichenbacher’s 1998 padding oracle attack is almost old enough to drink
    • ROBOT just won a Pwnie at Black Hat this week
PASETO – Rules for All Versions

- Formal specification regarding key/nonce-gen:
  - Just use urandom. Userspace PRNGs are forbidden. No more Mersenne Twister or LCGs.

- No IND-CCA2 insecure public key cryptography
  - PKCS #1 v1.5 is explicitly forbidden, forever
    - Bleichenbacher’s 1998 padding oracle attack is almost old enough to drink
    - ROBOT just won a Pwnie at Black Hat this week

- When possible, do everything in constant-time
PASETO – Rules for All Versions

• Formal specification regarding key/nonce-gen:
  – Just use urandom. Userspace PRNGs are forbidden. No more Mersenne Twister or LCGs.

• No IND-CCA2 insecure public key cryptography
  – PKCS #1 v1.5 is explicitly forbidden, forever
    • Bleichenbacher’s 1998 padding oracle attack is almost old enough to drink
    • ROBOT just won a Pwnie at Black Hat this week

• When possible, do everything in constant-time
  – Including base64url encoding
Pre-Authentication Encoding

- Prevents canonicalization attacks by ensuring unique inputs
Pre-Authentication Encoding

- Prevents canonicalization attacks by ensuring unique inputs
- Packs an array of strings into a string
Pre-Authentication Encoding

- Prevents canonicalization attacks by ensuring unique inputs
- Packs an array of strings into a string
- Prefix with the count of the number of pieces
Pre-Authentication Encoding

- Prevents canonicalization attacks by ensuring unique inputs
- Packs an array of strings into a string
- Prefix with the count of the number of pieces
- Each piece is prefixed with the length of the piece
Pre-Authentication Encoding

- Prevents canonicalization attacks by ensuring unique inputs
- Packs an array of strings into a string
- Prefix with the count of the number of pieces
- Each piece is prefixed with the length of the piece
- All integers are treated as unsigned 64-bit, little endian
Pre-Authentication Encoding

- Prevents canonicalization attacks by ensuring unique inputs
- Packs an array of strings into a string
- Prefix with the count of the number of pieces
- Each piece is prefixed with the length of the piece
- All integers are treated as unsigned 64-bit, little endian

\[ \text{PAE}(["test"]) => \x01\x00\x00\x00\x00\x00\x00\x04\x00\x00\x00\x00\x00\x00\x00\text{test} \]
PAE in Practice

• Version 1
  – Local
    • The HMAC-SHA384 tag appended to the ciphertext covers PAE([“v1.local.”, nonce, ciphertext, footer])
PAE in Practice

• Version 1
  - Local
    • The HMAC-SHA384 tag appended to the ciphertext covers PAE([“v1.local.”, nonce, ciphertext, footer])
  - Public
    • The message input for the RSA signature is PAE([“v1.public.”, message, footer])
PAE in Practice

• Version 2
  – Local
  • The additional data parameter for libsodium’s crypto_aead_xchacha20poly1305_encrypt() is PAE([“v2.local.”, nonce, footer])
PAE in Practice

• Version 2
  - Local
    • The additional data parameter for libsodium’s crypto_aead_xchacha20poly1305_encrypt() is PAE([“v2.local.”, nonce, footer])
    • Libsodium already includes the ciphertext in the Poly1305 authentication tag
PAE in Practice

• Version 2
  – Local
    • The additional data parameter for libsodium’s crypto_aead_xchacha20poly1305_encrypt() is PAE(“v2.local.”, nonce, footer)
    • Libsodium already includes the ciphertext in the Poly1305 authentication tag
  – Public
    • The message input for the Ed25519 signature is PAE(“v2.public.”, message, footer).
JWT vs PASETO

• JWT
  - Plethora of knobs and levers
  - Unauthenticated modes available
  - Promotes Reasoning by Lego
  - Often abused for stateless sessions

• PASETO
JWT vs PASETO

• JWT
  - Plethora of knobs and levers
  - Unauthenticated modes available
  - Promotes Reasoning by Lego
  - Often abused for stateless sessions

• PASETO
  - Only two options:
    • Version
    • Purpose
  - Everything is authenticated
    • Local-only tokens are also encrypted
  - Does its job, gets out of the way
To learn more about PASETO, visit:
https://paseto.io
https://github.com/paragonie/paseto
Designing Cryptography for Humans

• Opinionated interfaces with few options:
Designing Cryptography for Humans

• Opinionated interfaces with few options:
  - `encrypt(message, key[, ad = null])`
  - `decrypt(ciphertext, key[, ad = null])`
Designing Cryptography for Humans

• Opinionated interfaces with few options:
  – encrypt(message, key[, ad = null])
  – decrypt(ciphertext, key[, ad = null])

• Your users shouldn’t ever need to even *know* what a nonce is to encrypt safely
Designing Cryptography for Humans

- Opinionated interfaces with few options:
  - encrypt(message, key[, ad = null])
  - decrypt(ciphertext, key[, ad = null])
- Your users shouldn’t ever need to even know what a nonce is to encrypt safely
- Versioned protocols with hard-coded cipher-suites, vetted by cryptographers
Designing Cryptography for Humans

• Opinionated interfaces with few options:
  – encrypt(message, key[, ad = null])
  – decrypt(ciphertext, key[, ad = null])

• Your users shouldn’t ever need to even know what a nonce is to encrypt safely

• Versioned protocols with hard-coded cipher-suites, vetted by cryptographers
  – If a vulnerability is found in the current version, publish a new version with a better hard-coded ciphersuite
Cryptography for Mere Mortals, cont’d.

- Don’t just use simple binary strings for cryptography keys. Encapsulate them in a Key object.
Don’t just use simple binary strings for cryptography keys. Encapsulate them in a Key object.

- This discourages the use of human-sourced passwords as a cryptography key, without the added steps of a secure KDF function (Argon2)
Cryptography for Mere Mortals, cont’d.

- Don’t just use simple binary strings for cryptography keys. Encapsulate them in a Key object.
  - This discourages the use of human-sourced passwords as a cryptography key, without the added steps of a secure KDF function (Argon2)
  - In many languages, this also prevents keys from leaking into stack traces and ending up in JIRA/Trac tickets
Cryptography for Mere Mortals, cont’d.

- Logically separate symmetric cryptography from asymmetric cryptography
Cryptography for Mere Mortals, cont’d.

- Logically separate symmetric cryptography from asymmetric cryptography
  - javax.crypto.Cipher considered harmful
Cryptography for Mere Mortals, cont’d.

- Logically separate symmetric cryptography from asymmetric cryptography
  - javax.crypto.Cipher considered harmful
- Enforce failure modes through exceptions rather than returning null or false.
Cryptography for Mere Mortals, cont’d.

• Logically separate symmetric cryptography from asymmetric cryptography
  – `javax.crypto.Cipher` considered harmful
• Enforce failure modes through exceptions rather than returning null or false.
  – If the developer doesn’t catch the exception, your code fails closed. If they do, they can handle failure gracefully in a way that doesn’t seem like crashing.
Cryptography for Mere Mortals, cont’d.

- Logically separate symmetric cryptography from asymmetric cryptography
  - javax.crypto.Cipher considered harmful

- Enforce failure modes through exceptions rather than returning null or false.
  - If the developer doesn’t catch the exception, your code fails closed. If they do, they can handle failure gracefully in a way that doesn’t seem like crashing.
  - The alternatives (unavoidable crash, fail open) are bad. One scares developers, the other creates security holes in production systems.
Recap

- The blame game doesn’t solve insecurity
Recap

• The blame game doesn’t solve insecurity
• Prevent developers from rolling their own crypto by giving them tools that are hard to misuse
Recap

- The blame game doesn’t solve insecurity
- Prevent developers from rolling their own crypto by giving them tools that are hard to misuse
- Your API should be simple to understand
Recap

- The blame game doesn’t solve insecurity
- Prevent developers from rolling their own crypto by giving them tools that are hard to misuse
- Your API should be simple to understand
  - Every asterisk is a disaster risk
Recap

- The blame game doesn’t solve insecurity
- Prevent developers from rolling their own crypto by giving them tools that are hard to misuse
- Your API should be simple to understand
  - Every asterisk is a disaster risk
- Prefer versioned protocols over cipher agility
Recap

- The blame game doesn’t solve insecurity
- Prevent developers from rolling their own crypto by giving them tools that are hard to misuse
- Your API should be simple to understand
  - Every asterisk is a disaster risk
- Prefer versioned protocols over cipher agility
- Error-prone standards (JOSE) should be avoided in favor of safer designs (PASETO)
Questions?
Scott Arciszewski

- **Paragon Initiative Enterprises, LLC**
  - Software development (open source)
    - The person to blame for getting libsodium into PHP 7.2
    - Also wrote the sodium_compat polyfill for PHP 5.2 – 7.1
    - Many PHP security libraries
  - Security research
    - Handfuls of CVEs
    - Sometimes published on Full Disclosure
- **Twitter handle:** @CiPHPPerCoder