IMPLEMENTATION OF A FIDO2 AUTHENTICATOR LIBRARY IN ZIG

ABOUT

- David
- Student and academic staff at Aalen University
- **ABSOLUTELY NO** affiliation with the FIDO Alliance or any other organisation

AUTHENTICATION

- **Knowledge**: Something (only) you know
- **Posession**: Something you own
- **Inherence**: Something inherent to you

Passwords/ PINs are up to this day the most popular authentication method

- OS user accounts
- Credit Cards
- Web-services
- Access control systems

PROBLEMS

- Passwords must be increasingly complex
- This makes them hard to remember
- Leaked password hashes might be cracked
- Passwords might end up in log-files
- Users might fall for social-engineering/ fishing

Second factors (e.g. OTPs) can help mitigate attacks but they can be bypassed, e.g. using social-engineering

[Uber]

Every extra step included into a authentication procedure possibly worsens the user experience

Every extra step included into a authentication procedure possibly worsens the user experience

Authentication should be quick and painless

REQUIREMENTS

REQUIREMENTS

- Users shouldnt be responsible for details like password complexity

REQUIREMENTS

- Users shouldnt be responsible for details like password complexity
- Authentication should be fast ...

REQUIREMENTS

- Users shouldnt be responsible for details like password complexity
- Authentication should be fast ...
- ... and secure

REQUIREMENTS

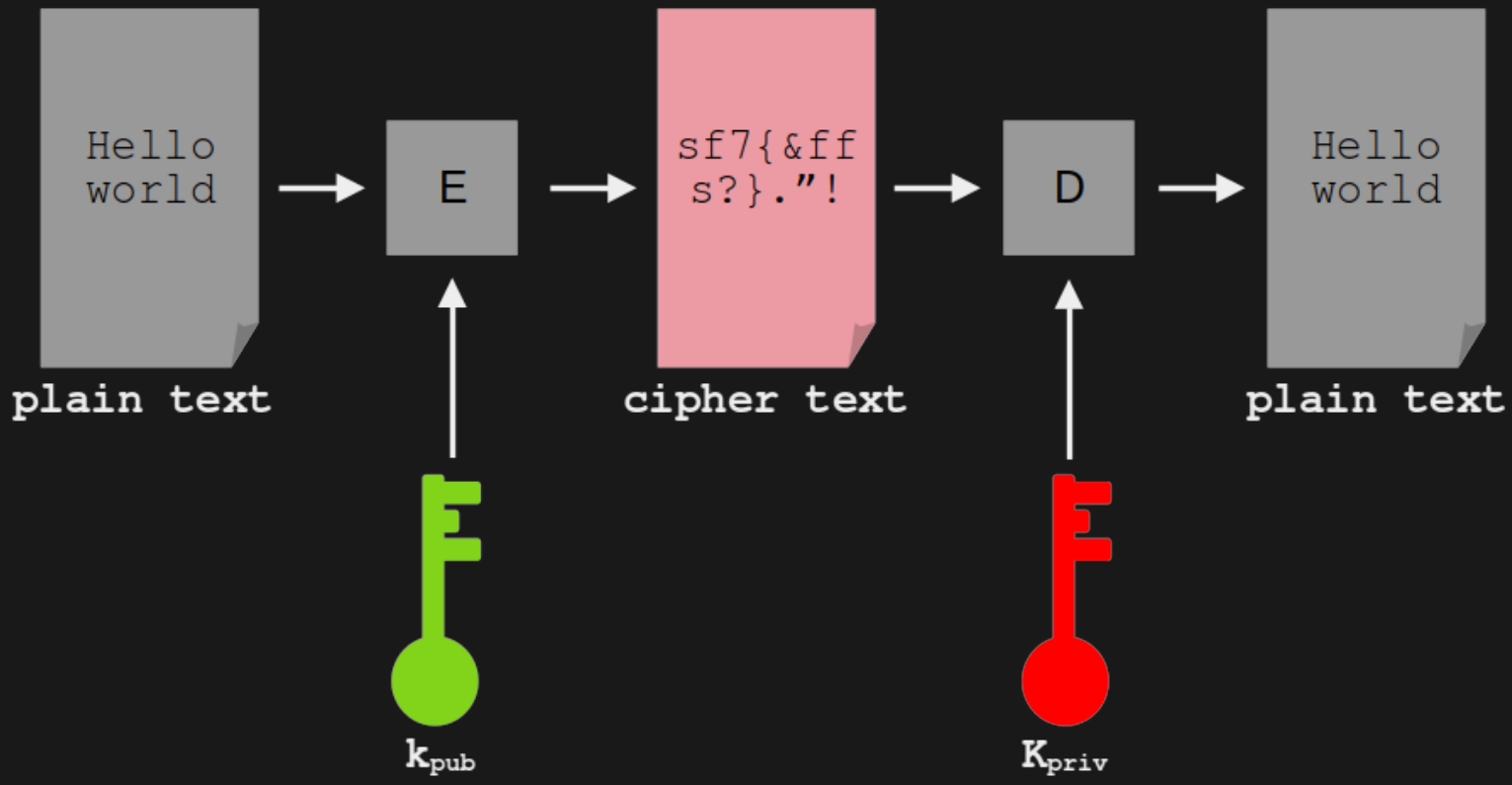
- Users shouldnt be responsible for details like password complexity
- Authentication should be fast ...
- ... and secure

BETTER SOLUTIONS? ...

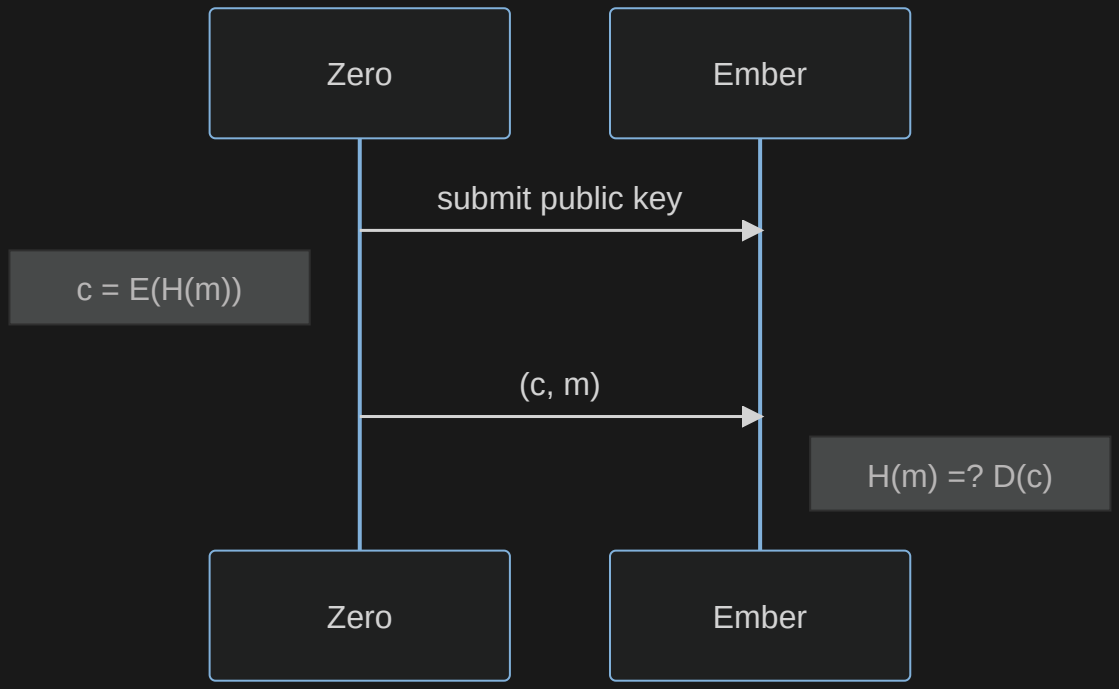
The FIDO Alliance developed FIDO Authentication standards based on public key cryptography for authentication that is more secure than passwords and SMS OTPs, simpler for consumers to use, and easier for service providers to deploy and manage. FIDO Authentication enables password-only logins to be replaced with secure and fast login experiences across websites and apps

[FIDO Alliance]

PUBLIC-KEY CRYPTOGRAPHY

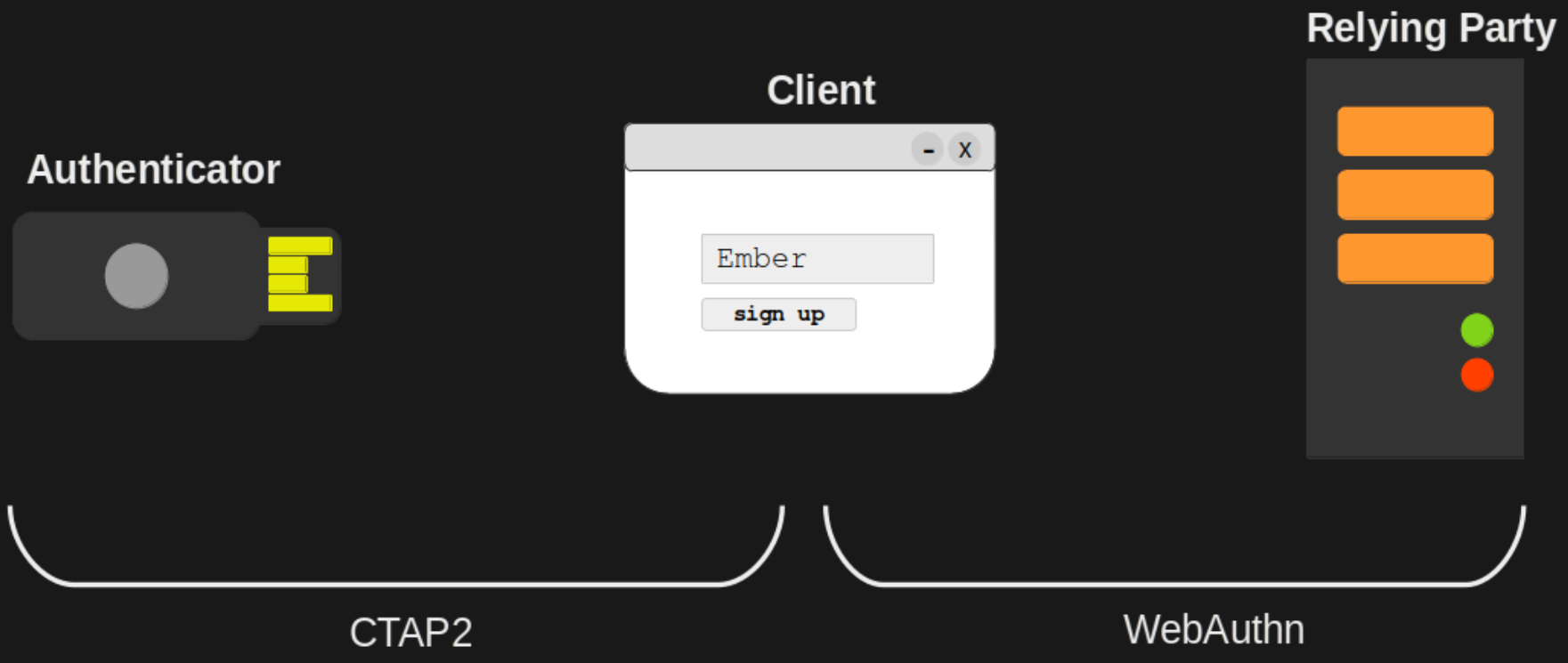

Hello world
E
sf7{&ff s?}.”!
D
Hello world
plain text
cipher text
plain text
k_{pub}
K_{priv}

Encrypting (the hash of) a message m using a private key k_{priv} is referred to as making a signature

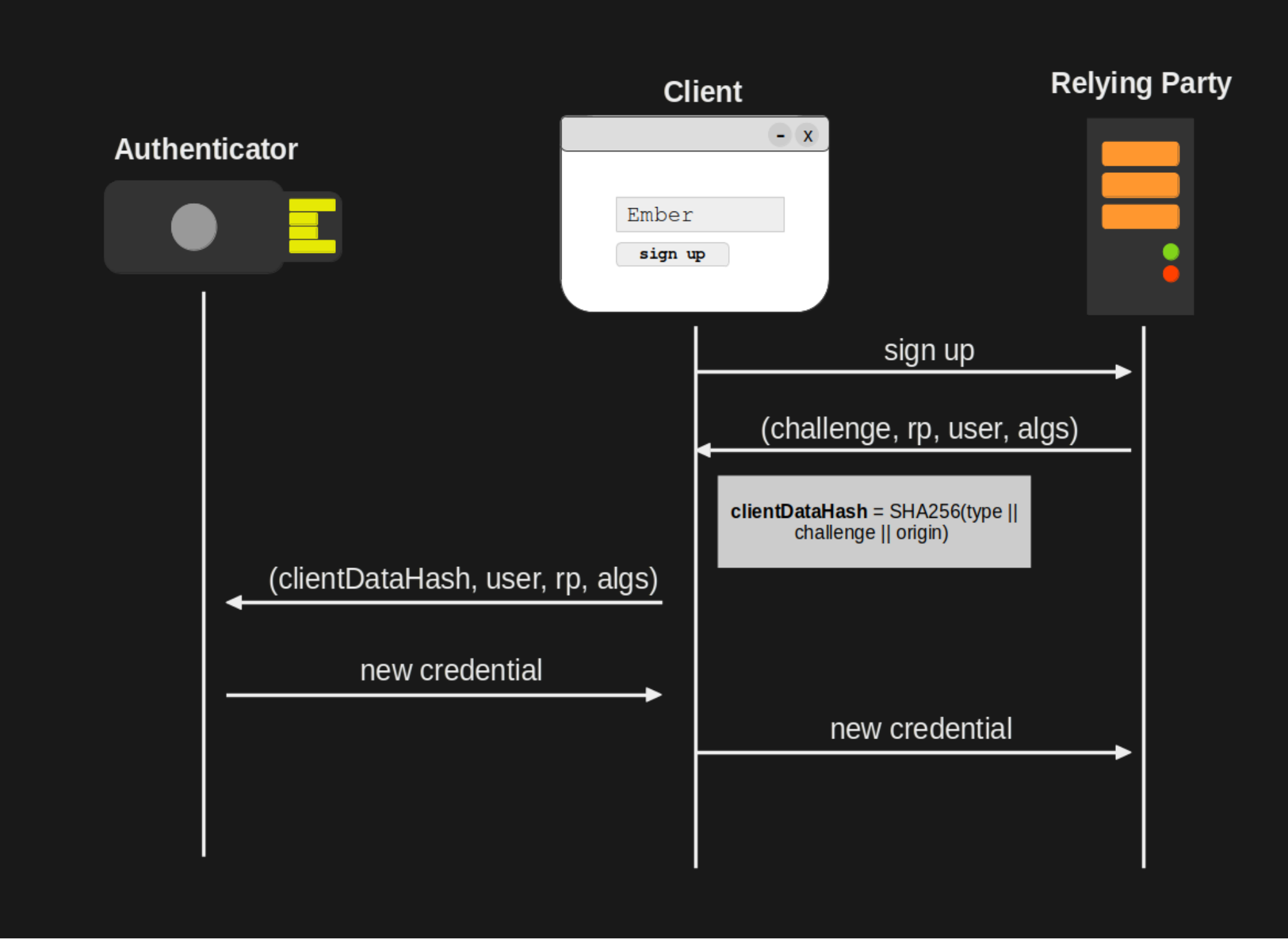

Zero
Ember
submit public key
c = E(H(m))
(c, m)
H(m) =? D(c)
Zero
Ember

FIDO2

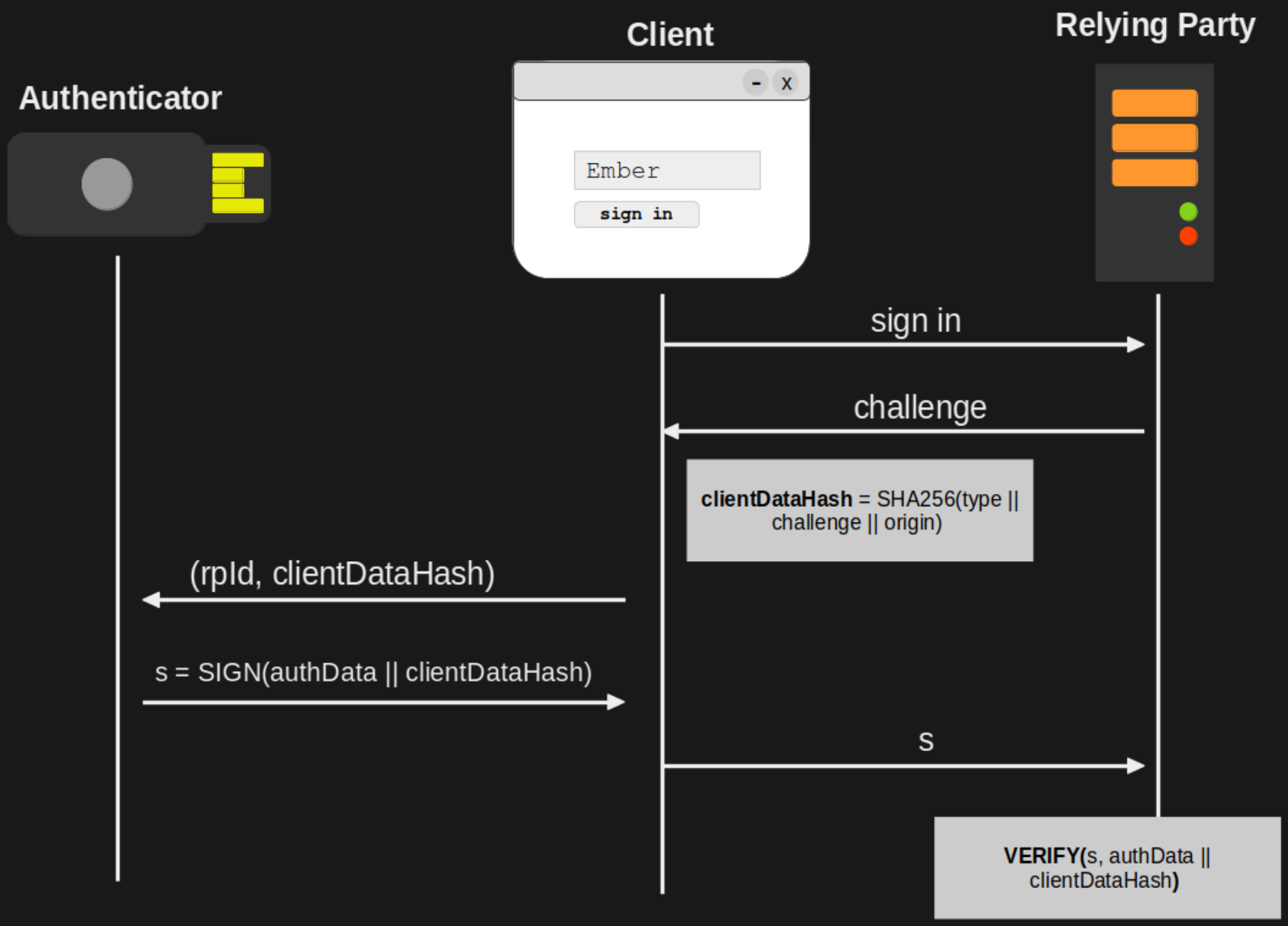
FIDO2 uses digital signatures for authentication and consists of two sub-protocols, CTAP2 and WebAuthn


Relying Party
Client
Authenticator
Ember
sign up
CTAP2
WebAuthn

REGISTRATION


Client
Relying Party
Authenticator
Ember
sign up
sign up
(challenge, rp, user, algs)
clientDataHash = SHA256(type || challenge || origin)
(clientDataHash, user, rp, algs)
new credential
new credential

AUTHENTICATION


Client
Relying Party
Authenticator
- x
Ember
sign in
sign in
challenge
clientDataHash = SHA256(type || challenge || origin)
(rpId, clientDataHash)
s = SIGN(authData || clientDataHash)
s
VERIFY(s, authData || clientDataHash)

ADVANTAGES

- Uses unique private keys bound to a relying party
- Only the public key is stored server-side
- Easy to use (if implemented correctly)
- Authenticators may be additionally protected by a PIN or built in user verification method

FIDO2 AUTHENTICATOR

Must implement the CTAP2 protocol

APPLICATION STRUCTURE

- Application layer: The API for creating new credentials and generating assertions

APPLICATION STRUCTURE

- Application layer: The API for creating new credentials and generating assertions
- Encoding layer: Data is encoded using the Concise Binary Object Representation (CBOR)

APPLICATION STRUCTURE

- Application layer: The API for creating new credentials and generating assertions
- Encoding layer: Data is encoded using the Concise Binary Object Representation (CBOR)
- Transport layer: Inter process communication, e.g. CTAPHID for USB

CTAP2

Layer			PDU [4]	Function
Host	4	Application	Data	Creation of credentials and assertions
	3	Encoding	Data (CBOR)	Translation of data from/to CBOR
	2	Transport	Messages, Packets	Reliable transmission of messages between client and authenticator (CTAPHID)
Media	1	Physical	Bit	Transmission and reception of raw bit streams over a physical medium

AUTHENTICATOR LIBRARY

- Provides CTAP2 API calls (e.g. authenticatorMakeCredential, authenticatorGetAssertion)
- Allows encoding/ decoding of CBOR messages
- Provides protocols for the different transports (e.g. CTAPHID for USB)
- The user provides callbacks for the interaction with the system (e.g. microcontroller, OS, ...)

Application and Encoding

```
var authenticator = fido.ctap.authenticator.Authenticator{
    .settings = .{
        .versions = &.{ .FIDO_2_0, .FIDO_2_1 },
        .aaguid = "\x7f\x15\x82\x74\xaa\xb6\x44\x3d\x9b\xcf\x
        .options = .{
            .plat = true,
            .clientPin = true,
            .pinUvAuthToken = true,
        },
        .pinUvAuthProtocols = &.{.V2},
        .transports = &.{.usb},
        .algorithms = &.{.{ .alg = .Es256 }},
        .firmwareVersion = 0xcafe,
    },
    .attestation_type = .Self,
```

Application and Encoding

```
15    .attestation_type = .Self,
16    .callbacks = .{
17        .rand = callbacks.rand,
18        .millis = callbacks.millis,
19        .up = callbacks.up,
20        .loadCurrentStoredPIN = callbacks.loadCurrentStoredPI
21        .storeCurrentStoredPIN = callbacks.storeCurrentStored
22        .loadPINCodePointLength = callbacks.loadPINCodePointL
23        .storePINCodePointLength = callbacks.storePINCodePoin
24        .get_retries = callbacks.get_retries,
25        .set_retries = callbacks.set_retries,
26        .load_credential_by_id = callbacks.load_credential_by
27        .store_credential_by_id = callbacks.store_credential_
28    },
29    .token = .{
```

Application and Encoding

```
19          .up = callbacks.up,
20          .loadCurrentStoredPIN = callbacks.loadCurrentStoredPI
21          .storeCurrentStoredPIN = callbacks.storeCurrentStored
22          .loadPINCodePointLength = callbacks.loadPINCodePointL
23          .storePINCodePointLength = callbacks.storePINCodePoin
24          .get_retries = callbacks.get_retries,
25          .set_retries = callbacks.set_retries,
26          .load_credential_by_id = callbacks.load_credential_by
27          .store_credential_by_id = callbacks.store_credential_
28      },
29      .token = .{
30          .two = fido.ctap.pinuv.PinUvAuth.v2(callbacks.rand),
31      },
32      .allocator = gpa.allocator(),
33  };
```

Transport

```
while (true) {
    const msg = try usb.read();

    var response = fido.ctap.transports.ctaphid.authenticat
        msg,
        &authenticator,
    );

    if (response) |*resp| {
        while (resp.next()) |packet| {
            try usb.write(packet);
        }
    }
}
```

Transport

```
while (true) {
    const msg = try usb.read();

    var response = fido.ctap.transports.ctaphid.authenticat
        msg,
        &authenticator,
    );

    if (response) |*resp| {
        while (resp.next()) |packet| {
            try usb.write(packet);
        }
    }
}
```

Transport

```
while (true) {
    const msg = try usb.read();

    var response = fido.ctap.transports.ctaphid.authenticat
        msg,
        &authenticator,
    );

    if (response) |*resp| {
        while (resp.next()) |packet| {
            try usb.write(packet);
        }
    }
}
```

WHY ZIG?

Zig already provides a solid foundation of cryptographic algorithms in its standard library

- Signature algorithms
- Hashing
- Encryption

Zig already provides a solid foundation of cryptographic algorithms in its standard library

- Signature algorithms
- Hashing
- Encryption
- Support for different elliptic curves

```
pub const EcdhP256 = Ecdh(crypto.ecc.P256);

pub fn Ecdh(comptime Curve: type) type {
    return struct {
        pub const secret_length = Curve.scalar.encoded_leng
        pub const public_length = Curve.Fe.encoded_length;

        // ...

        pub fn scalarmultXY(secret_key: [secret_length]u8,
            const x = try Curve.Fe.fromBytes(pub_x[0..].*,
            const y = try Curve.Fe.fromBytes(pub_y[0..].*,
            const c = try Curve.fromAffineCoordinates(.{ .x
            const secret = try c.mul(secret_key, .Little);
            return secret;
```

```
4       return struct {
5           pub const secret_length = Curve.scalar.encoded_leng
6           pub const public_length = Curve.Fe.encoded_length;
7
8           // ...
9
10          pub fn scalarmultXY(secret_key: [secret_length]u8,
11              const x = try Curve.Fe.fromBytes(pub_x[0..].*,
12              const y = try Curve.Fe.fromBytes(pub_y[0..].*,
13              const c = try Curve.fromAffineCoordinates(.{ .x
14              const secret = try c.mul(secret_key, .Little);
15              return secret;
16          }
17      };
18  }
```

Support for different architectures and operating systems

- nRF52840 MDK USB Dongle
- Linux

Easy to generate code at compile time

```
pub fn parse(
    /// The type to deserialize to
    comptime T: type,
    /// The data item to deserialize
    item: DataItem,
    /// Options to effect the behaviour of this function
    options: ParseOptions,
) ParseError!T {
    switch (@typeInfo(T)) {
        .Bool => {
            return switch (item.getType()) {
                .False => false,
                .True => true,
                else => ParseError.UnexpectedItem,
            };
        },
        .Float, .ComptimeFloat => {
            return switch (item.getType()) {
                .Float => if (item.float()) |x| @floatCast(T, x) else return ParseError.Malformed,
                else => ParseError.UnexpectedItem,
            };
        },
        .Int, .ComptimeInt => {
            switch (item.getType()) {
                .Int => {
                    const v = if (item.int()) |x| x else return ParseError.Malformed;
```

Easy to generate code at compile time

```
pub fn parse(
    /// The type to deserialize to
    comptime T: type,
    /// The data item to deserialize
    item: DataItem,
    /// Options to effect the behaviour of this function
    options: ParseOptions,
) ParseError!T {
    switch (@typeInfo(T)) {
        .Bool => {
            return switch (item.getType()) {
                .False => false,
                .True => true,
                else => ParseError.UnexpectedItem,
            };
        },
        .Float, .ComptimeFloat => {
            return switch (item.getType()) {
                .Float => if (item.float()) |x| @floatCast(T, x) else return ParseError.Malformed,
                else => ParseError.UnexpectedItem,
            };
        },
        .Int, .ComptimeInt => {
            switch (item.getType()) {
                .Int => {
                    const v = if (item.int()) |x| x else return ParseError.Malformed;
```

Easy to generate code at compile time

```
pub fn parse(
    /// The type to deserialize to
    comptime T: type,
    /// The data item to deserialize
    item: DataItem,
    /// Options to effect the behaviour of this function
    options: ParseOptions,
) ParseError!T {
    switch (@typeInfo(T)) {
        .Bool => {
            return switch (item.getType()) {
                .False => false,
                .True => true,
                else => ParseError.UnexpectedItem,
            };
        },
        .Float, .ComptimeFloat => {
            return switch (item.getType()) {
                .Float => if (item.float()) |x| @floatCast(T, x) else return ParseError.Malformed,
                else => ParseError.UnexpectedItem,
            };
        },
        .Int, .ComptimeInt => {
            switch (item.getType()) {
                .Int => {
                    const v = if (item.int()) |x| x else return ParseError.Malformed;
```

Easy to generate code at compile time

```
pub fn parse(
    /// The type to deserialize to
    comptime T: type,
    /// The data item to deserialize
    item: DataItem,
    /// Options to effect the behaviour of this function
    options: ParseOptions,
) ParseError!T {
    switch (@typeInfo(T)) {
        .Bool => {
            return switch (item.getType()) {
                .False => false,
                .True => true,
                else => ParseError.UnexpectedItem,
            };
        },
        .Float, .ComptimeFloat => {
            return switch (item.getType()) {
                .Float => if (item.float()) |x| @floatCast(T, x) else return ParseError.Malformed,
                else => ParseError.UnexpectedItem,
            };
        },
        .Int, .ComptimeInt => {
            switch (item.getType()) {
                .Int => {
                    const v = if (item.int()) |x| x else return ParseError.Malformed;
```

Easy to generate code at compile time

```
pub fn parse(
    /// The type to deserialize to
    comptime T: type,
    /// The data item to deserialize
    item: DataItem,
    /// Options to effect the behaviour of this function
    options: ParseOptions,
) ParseError!T {
    switch (@typeInfo(T)) {
        .Bool => {
            return switch (item.getType()) {
                .False => false,
                .True => true,
                else => ParseError.UnexpectedItem,
            };
        },
        .Float, .ComptimeFloat => {
            return switch (item.getType()) {
                .Float => if (item.float()) |x| @floatCast(T, x) else return ParseError.Malformed,
                else => ParseError.UnexpectedItem,
            };
        },
        .Int, .ComptimeInt => {
            switch (item.getType()) {
                .Int => {
                    const v = if (item.int()) |x| x else return ParseError.Malformed;
```

Easy to generate code at compile time

```
                    if (cmp(field.name, k)) {
51                      return @field(T, field.name);
52                  }
53              }
54              return ParseError.InvalidEnumTag;
55          },
56          else => return ParseError.UnexpectedItem,
57      }
58  },
59  .Struct => |structInfo| {
60      // Custom parse function overrides default behaviour
61      const has_parse = comptime std.meta.trait.hasFn("cborParse")(T);
62      if (has_parse and !options.from_cborParse) {
63          return T.cborParse(item, options);
64      }
65
66      switch (item.getType()) {
67          .Map => {
68              var r: T = undefined;
69              var fields_seen = [_]bool{false} ** structInfo.fields.len;
70
71              var v = if (item.map()) |x| x else return ParseError.Malformed;
72              while (v.next()) |kv| {
73                  var found = false;
74
75                  if (kv.key.getType() != .TextString and kv.key.getType() != .Int) continue;
```

Easy to generate code at compile time

```
59          .Struct => |structInfo| {
60              // Custom parse function overrides default behaviour
61              const has_parse = comptime std.meta.trait.hasFn("cborParse")(T);
62              if (has_parse and !options.from_cborParse) {
63                  return T.cborParse(item, options);
64              }
65
66              switch (item.getType()) {
67                  .Map => {
68                      var r: T = undefined;
69                      var fields_seen = [_]bool{false} ** structInfo.fields.len;
70
71                      var v = if (item.map()) |x| x else return ParseError.Malformed;
72                      while (v.next()) |kv| {
73                          var found = false;
74
75                          if (kv.key.getType() != .TextString and kv.key.getType() != .Int) continue;
76
77                          inline for (structInfo.fields, 0..) |field, i| {
78                              var match: bool = false;
79                              var name: []const u8 = field.name;
80
81                              // Is there an alias specified?
82                              for (options.field_settings) |fs| {
83                                  if (std.mem.eql(u8, field.name, fs.name)) {
84                                      if (fs.alias) |alias| {
```

Easy to generate code at compile time

```
65
66              switch (item.getType()) {
67                  .Map => {
68                      var r: T = undefined;
69                      var fields_seen = [_]bool{false} ** structInfo.fields.len;
70
71                      var v = if (item.map()) |x| x else return ParseError.Malformed;
72                      while (v.next()) |kv| {
73                          var found = false;
74
75                          if (kv.key.getType() != .TextString and kv.key.getType() != .Int) continue;
76
77                          inline for (structInfo.fields, 0..) |field, i| {
78                              var match: bool = false;
79                              var name: []const u8 = field.name;
80
81                              // Is there an alias specified?
82                              for (options.field_settings) |fs| {
83                                  if (std.mem.eql(u8, field.name, fs.name)) {
84                                      if (fs.alias) |alias| {
85                                          name = alias;
86                                      }
87                                  }
88                              }
89
90                              switch (kv.key.getType()) {
```

Easy to generate code at compile time

```
83                          if (std.mem.eql(u8, field.name, fs.name)) {
84                              if (fs.alias) |alias| {
85                                  name = alias;
86                              }
87                          }
88                      }
89
90                      switch (kv.key.getType()) {
91                          .Int => {
92                              const x = if (kv.key.int()) |y| y else return ParseError.Malformed;
93                              const y = s2n(name);
94                              match = x == y;
95                          },
96                          else => {
97                              match = std.mem.eql(u8, name, if (kv.key.string()) |x| x else return
98                          },
99                      }
100
101                     if (match) {
102                         if (fields_seen[i]) {
103                             switch (options.duplicate_field_behavior) {
104                                 .UseFirst => {
105                                     found = true;
106                                     break;
107                                 },
108                                 .Error => return ParseError.DuplicateCborField,
```

Easy to generate code at compile time

```
                    switch (options.duplicate_field_behavior) {
                        .UseFirst => {
                            found = true;
                            break;
                        },
                        .Error => return ParseError.DuplicateCborField,
                    }
                }

                var child_options = options;
                child_options.from_cborParse = false;
                @field(r, field.name) = try parse(
                    field.type,
                    kv.value,
                    child_options,
                );
                errdefer {
                    // TODO: add error defer to free memory
                    const I = @typeInfo(@TypeOf(@field(r, field.name)));

                    switch (I) {
                        .Pointer => |ptrInfo| {
                            _ = ptrInfo;
                        },
                        else => {},
                    }
                }
```

Easy to generate code at compile time

```
cbor.parse(MakeCredential, cbor.DatItem.new(buffer[0..]), .{
    .allocator = self.allocator,
})
```

CLOSING THOUGHTS

- Authentication is a central part of our online activities
- Unfortunately there are many pitfalls
- FIDO2 could be a better alternative, but its far away from widespread adoption
  - Hardware authenticators are expensive
  - Missing IPC spec for platform authenticators
  - Some browsers, and a lot of websites lack support

WHAT'S MISSING IS A (PLATFORM-)AUTHENTICATOR YOU CAN LOVE ;)

- https://github.com/r4gus/fido2

QUESTIONS ?