# FIDOU2F技术文档

## 关于U2F

Universal 2ndFactor—双因子认证。

U2F是在现有的用户名+密码认证的基础之上，增加一个更安全的认证因子用于登录认证。用户可以像以前一样通过用户名和密码登录服务，服务会提示用户出示一个第二因子设备来进行认证。U2F可以使用简单的密码（比如4个数字的PIN）而不牺牲安全性。

## 2. 工作流程

### 2.1 U2F 注册流程

注册流程一：请求注册安全秘钥

1、用户登录成功，在帐户设置设置界面选择添加安全秘钥设备；

2、服务器验证此帐户信息合法性以及该帐户下已经注册过的安全秘钥设备；

3、生成注册请求数据下发至设备；

RegisterRequestData {

String appId; //域名

List<RegisteredKey> registeredKeys; //已注册的安全秘钥设备keyHandle

List<RegisterRequest> registerRequests; //挑战值Challenge（RequestID）

}

注册流程二：安全秘钥数据返回

1、浏览器获取到RegisterRequestData注册请求数据后通过u2f-api.js下发至安全秘钥设备；

2、设备判断此帐户是否已经注册本身或者返回数据并提交至服务器；

RegisterResponse{

String RegistrationData; //注册数据

String ClientDataRaw; //原始Client数据

ClientData clientDataRef { //Client对象引用数据

String type;

String challenge;

String origin;

}

}

注册流程三：服务器数据校验

1、查询出注册请求RegisterRequest并获取挑战值：RegisterRequest.getChallenge()；

2、从RegisterResponse中获取原始注册数据：

RawRegisterResponse{

Crypto crypto; //加解密工具

byte[] userPublicKey； //用户公钥

byte[] keyHandle； //注册唯一标识

X509Certificate attestationCertificate; //设备证书

byte[] signature； //签名

}

3、签名验证checkSignature；

注册流程四：安全秘钥注册成功

1、创建安全秘钥设备注册信息并初始化counter；

DeviceRegistration{

String keyHandle; //生成的KeyHandle

String publicKey; //用户公钥

String attestationCert; //设备证书

long counter; //设备验证次数

boolean compromised; //设备被盗用标志

}

2、判断此设备是否允许注册；根据安全秘钥设备证书信息在服务器MDS中查询可认可的设备元数据信息；

3、将安全秘钥设备信息存储并关联帐户KeyHandle和PublicKey,

### 2.2 U2F认证流程

验证流程一：验证安全秘钥请求

1、用户登录成功，插入安全设备进行验证；

2、服务器验证此帐户信息合法性以及该帐户下已注册过的安全秘钥设备的合法性；

No devices registrered；All devices compromised；

生成注册请求：

SignRequestData{

String appId; //域名

String challenge; //挑战值

List<SignRequest> signRequests{ //验证请求集合

String challenge; //挑战值

String appId; //域名

String keyHandle; //服务器查询的Keyhandle

}

}

验证流程二：安全秘钥数据返回

1、浏览器获取到SignRequestData注册请求数据后通过u2f-api.js下发至安全秘钥设备；

2、设备判断此帐户是否已经注册本身或者返回数据并提交至服务器；

SignResponse{

String clientDataRaw; //原始Client数据

ClientData clientDataRef; //Client对象引用数据

String signatureData; //签名数据

String keyHandle; //返回的KeyHandle

}

验证流程三：服务器数据校验与验证成功

1、检测设备是否存在在帐户已经注册的设备列表中；

2、检测设备是否被盗用以及与提交数据中keyHandle是否一致；

3、从SignResponse中获取原始验证数据：

RawSignResponse{

byte userPresence; //用户是否进行了存在性校验

long counter; //验证次数

byte[] signature; //数据签名

final Crypto crypto; //签名算法

}

4、对原始验证数据做签名检查；

5、检测确认用户存在校验值是否一致；

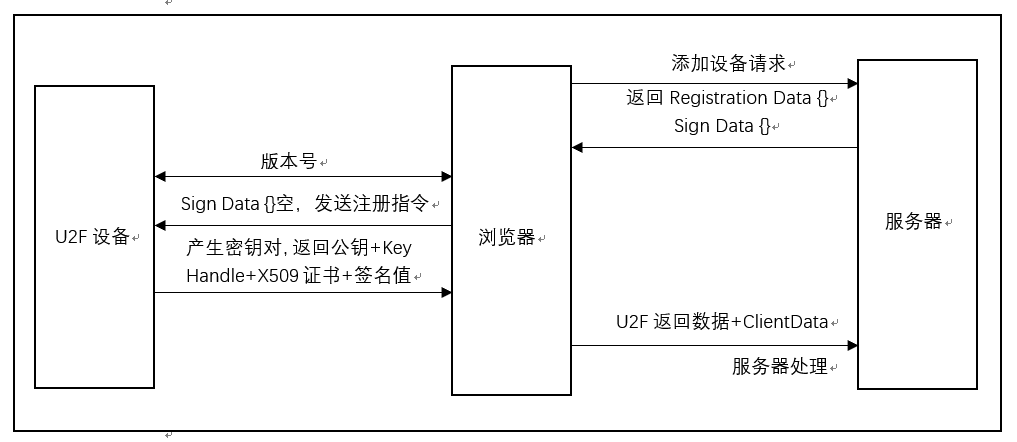
6、检测计数器counter是否合格；

## 3．U2F 协议解析

### 3.1服务器关联的密钥对

U2F设备产生的密钥对应该是服务器关联的，一对密钥对应一个服务器，而不是一个U2F设备对应一个服务器。在注册的时候，服务器给U2F设备传入服务器相关信息，U2F设备产生一对密钥对，将此密钥对和服务器相关信息相关联，给此密钥对分配一个句柄（Key Handle），并将其和公钥传给服务器，服务器将注册的账户信息，公钥，Key Handle全部关联在一起并保存。

当用户需要使用U2F做验证操作时，服务器产生挑战数据，使用U2F设备做签名，此时服务器将Key Handle和服务器信息通过浏览器传给U2F设备，U2F设备使用Key Handle，寻找对应的密钥对，如果密钥对存在，检验密钥对应的服务器信息是否和传入的服务器信息匹配，如果不匹配，说明服务器是伪造或者不正确的。如果正确，U2F设备等待用户按键确认，用户按键后，U2F设备对挑战数据做签名，签名值返回给服务器，服务器验证签名值，如果签名正确，说明此公钥对应的唯一私钥是正确的，表面用户拥有合法U2F设备，如果签名不正确，说明此用户正在伪造身份登录。可见，U2F验证身份是双向的，U2F验证服务器的真伪，服务器验证U2F的真假。



添加设备请求

服务器验证，生成注册请求数据RegisterRequestData{}

版本号

产生密钥对，返回RegisterResponse

U2F返回数据+ClientData

服务器

浏览器

U2F设备

RegisterRequest.getChallenge()；RegisterResponse中获取原始注册数据；签名验证checkSignature

通过u2f-api.js下发至安全秘钥设备

认证请求

SignRequestData{}

验证Key Handle，Key Handle和 application id 是否匹配。如果验证成功，通过Key Handle对应的私钥对指定数据做签名，计数器加一，返回SignResponse

U2F返回数据+ClientData

服务器

浏览器

U2F设备

### ,

服务器从SignResponse中获取原始验证数据RawSignResponse{},签名检查,检测确认用户存在校验值是否一致,检测计数器counter是否合格。

