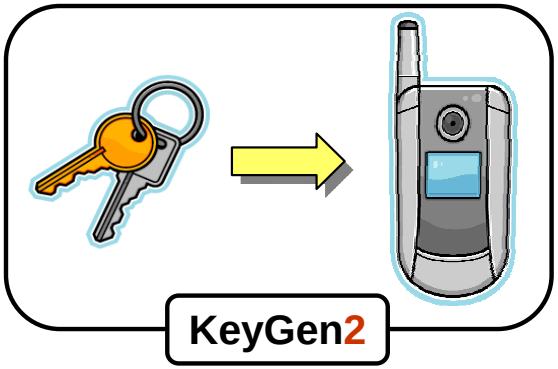


KeyGen2* – “Universal Provisioning”

KeyGen2 is an effort creating a standard for browser-based on-line provisioning of PKI user-certificates and keys.

In addition, KeyGen2 supports an option for “piggybacking” symmetric keys like OTP (One Time Password) seeds on PKI, which is a way of leveraging KeyGen2 as well as providing higher security, more sophisticated key-management facilities, and improved flexibility compared to most current symmetric-key-only provisioning systems.

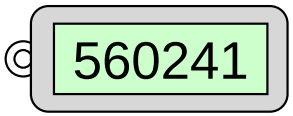
Using a generic credential extension mechanism, KeyGen2 can support things like Microsoft InfoCards and downloadable code associated with a specific key.

One of the core targets for KeyGen2 are mobile phones equipped with TPMs (Trusted Platform Modules), which properly applied, *can securely emulate any number of smart-cards*.

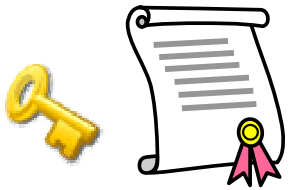
* KeyGen2 is a tribute to KeyGen which was the first browser-based PKI provisioning protocol, introduced by Netscape 1996.

KeyGen2 – Core Features

- Supports all the authentication and signature keys a *consumer*, *citizen*, or *employee* may need (*any organization* + *any technology*)
- Supports user-key lifecycle management operations ranging from semi-automatic renewals to credential policy updates
- Supports issuer-specific PIN-codes, policies, and PUKs
- Supports a Browser as well as a “Web Service” interface
- Builds on established Internet standards such as HTTPS, MIME, XML Schema, XML Signature, and XML Encryption
- Works equally well in a non-managed device as well as in a managed device
- Supports cryptographic containers vouching for generated keys through key-attestation signatures, which enable issuers verifying that keys actually reside in a "safe harbor" rather than in unknown territory



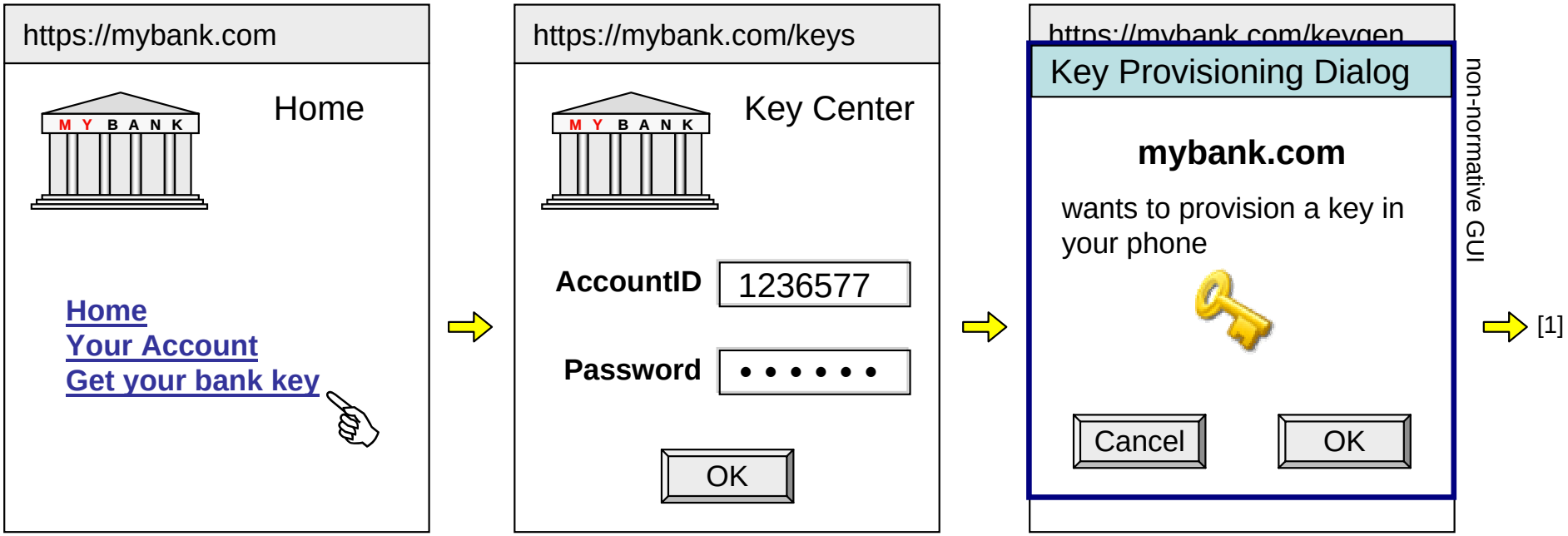
OTP (One Time Password)



PKI (Public Key Infrastructure)

Microsoft CardSpace

Browser-based Key Provisioning - 1/2



- Using a mobile browser makes the enrolment procedure and authentication method independent of the final key provisioning step
- A browser scheme is most suited for entities that do not manage the device
- A browser-based key provisioning system can be made compatible with traditional e-mail loop back address verification over the Internet as well as with strict procedures in a passport office using an NFC/Bluetooth connection to a local provisioning server

1] Additional key provisioning steps left out for brevity

Browser-based Key Provisioning - 2/2

Arbitrary GET or POST operation through the mobile browser

Response using the MIME-type `application/xbpp+xml`

Check if the returned data contains an XML object with the KeyGen2 namespace and a `PlatformNegotiationRequest` top element

Key Provisioning Dialog

mybank.com

wants to provision a key in your phone

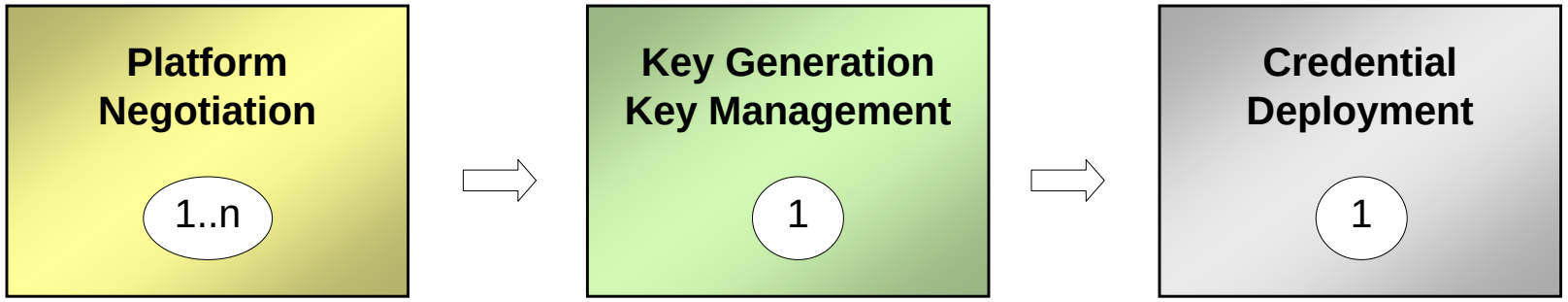
Cancel

OK

KeyGen2 protocol invocation!

The rest of the KeyGen2 protocol exchanges

KeyGen2 Protocol Phases



The number in the circle tells how many times the actual phase may occur. Platform Negotiation essentially deals with figuring out what cryptographic container to use as well as algorithm capabilities of the client-platform (like if it does it supports ECDSA).

The last phase, Credential Deployment may run as a separate task in the case there is a certification-period or similar between the Key Generation and the actual issuance of credentials.

KeyGen2 - Protocol Basics

Note: Platform Negotiation is currently TBD

In the beginning there was an empty key-store…

A key-issuer requests the owner (client) to “mint” a fresh RSA key-pair

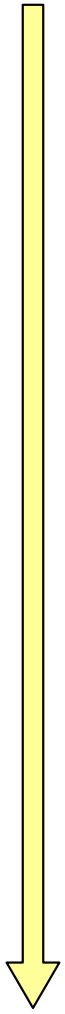
```
<KeyOperationRequest  ... >

   <CreateObject>

      <KeyPair ID="Key.1" KeyUsage="authentication">
         <RSA KeySize="2048"/>
      </KeyPair>

   </CreateObject>

</KeyOperationRequest>
```

The client's response

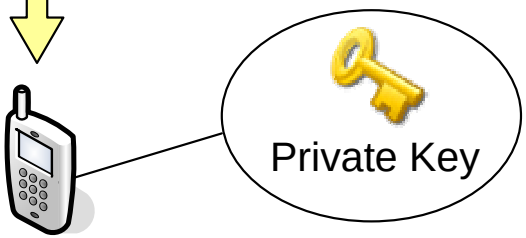
```
<KeyOperationResponse ... >
  <GeneratedPublicKey ID="Key.1">
    <ds:Signature>
      <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <ds:Reference URI="#Key.1">
          <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <ds:DigestValue>kqvBiyGe27B1twVFykLLuVq5kPs=</ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>TObiq9I….Mvj+oU=</ds:SignatureValue>
      <ds:KeyInfo>
        <ds:KeyValue>
          <ds:RSAKeyValue>
            <ds:Modulus>AKPRc….kESkV</ds:Modulus>
            <ds:Exponent>AQAB</ds:Exponent>
          </ds:RSAKeyValue>
        </ds:KeyValue>
      </ds:KeyInfo>
    </ds:Signature>
  </GeneratedPublicKey>

</KeyOperationResponse>
```

This is the generated RSA public key. The XML- signature functions as a "proof-of-possession" of the matching private key which only the client has access to. A conforming KeyGen2 implementation SHOULD also include a `Reference` to the `KeyInfo` object (i.e. signing the public key itself).

Install the certified key returned by the issuer in the key-store



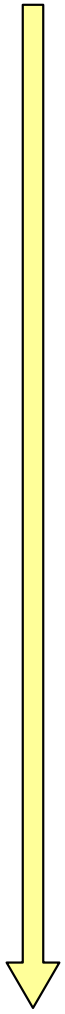
```
<CredentialDeploymentRequest ... >

   <CertifiedPublicKey ID="Key.1">
      <ds:X509Data>
         <ds:X509Certificate>MIIDnTCC…AoWgA=</ds:X509Certificate>
      </ds:X509Data>
   </CertifiedPublicKey>

</CredentialDeploymentRequest>
```



The private key and associated public key certificate are now ready to use!

KeyGen2 - Advanced Protocol Examples

Adding PIN-code protection and associated policies to a key



```
<KeyOperationRequest ... >

    <CreateObject>

      <PINPolicy Format="numeric" MaxLength="8" MinLength="5"
                 PatternRestrictions="three-in-a-row sequence" RetryLimit="3">

        <KeyPair ID="Key.1" KeyUsage="authentication">
          <RSA KeySize="2048"/>
        </KeyPair>

      </PINPolicy>

    </CreateObject>

</KeyOperationRequest>
```

The sample above only shows a part of the possible options regarding PIN-codes. You may also have the PIN preset as well as *enclosing multiple keys*. The policy specified in the sample would flag the following PINs as invalid:

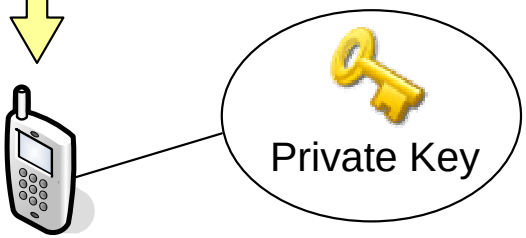
654321	Sequence
11123	Three in a row
1A567	Not numeric
4097	Too short

Provisioning an OTP “seed” or similar symmetric shared key

```
<CredentialDeploymentRequest ... >

  <CertifiedPublicKey ID="Key.1">
    <ds:X509Data>
      <ds:X509Certificate>MIIDnT … Yn4dPY=</ds:X509Certificate>
    </ds:X509Data>
    <PiggybackedSymmetricKey MAC="h54fg … 6LA3dj="
      EndorsedAlgorithms="http://www.w3.org/2000/09/xmldsig#hmac-sha1">
      <xenc:EncryptedKey>
        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        <xenc:CipherData>
          <xenc:CipherValue>IqKkjf2LuLL … 2JF8g+6Q=</xenc:CipherValue>
        </xenc:CipherData>
      </xenc:EncryptedKey>
    </PiggybackedSymmetricKey>
    <PropertyBag Type="urn:otpstd:spec">
      <Property Name="login" Value="janedoe"/>
      <Property Name="digits" Value="8"/>
      <Property Name="counter" Value="0" Writable="true"/>
    </PropertyBag>
  </CertifiedPublicKey>
```

Public Key
(X509v3 Cert)

Wrapped
Symmetric
Key



In order to fully provision a symmetric key, you now need to decrypt the wrapped symmetric key with the matching private key as well as storing possible additional key attribute-data (`Property` objects). To benefit from KeyGen2’s object-management scheme you also need to make a link between the “piggybacked” certificate and the symmetric key. `EndorsedAlgorithms` holds a list of issuer-endorsed algorithms.

Note: The X509 certificate functions as a universal key ID for all credential types.

Supporting Microsoft's Managed InfoCards


```
<CertifiedPublicKey ID="Key.1">
   <ds:X509Data>
      <ds:X509Certificate>MIIDnTCCA … WMYn4dPY=</ds:X509Certificate>
   </ds:X509Data>
   <Extension Type="http://schemas.microsoft.com/ws/2005/05/identity">PGRza ... 0dXJlPg0K</Extension>
</CertifiedPublicKey>
```

Base64-encoded InfoCard

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
   <Object Id="_Object_InfoCard">
      <InformationCard xml:lang="en-us"
                       xmlns="http://schemas.microsoft.com/ws/2005/05/identity">
         <UserCredential>
            <X509V3Credential>
               <X509Data xmlns="http://www.w3.org/2000/09/xmldsig#">
                  <KeyIdentifier
ValueType="http://docs.oasis-open.org/wss/2004/xx/oasis-2004xx-wss-soap-message-security-1.1#ThumbprintSHA1"
xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">V43s...Q=</KeyIdentifier>
               </X509Data>
            </X509V3Credential>
         </UserCredential>
         <SupportedTokenTypeList>
            <TokenType xmlns="http://schemas.xmlsoap.org/ws/2005/02/trust">urn:oasis:names:tc:SAML:1.0:assertion</TokenType>
         </SupportedTokenTypeList>
         <SupportedClaimTypeList>
            <SupportedClaimType Uri="http://schemas.microsoft.com/ws/2005/05/identity/claims/givenname">
               <DisplayTag>Given Name</DisplayTag>
            </SupportedClaimType>
         </SupportedClaimTypeList>
      </InformationCard>
   </Object>
</Signature>
```

A managed InfoCard using an X.509 credential for authentication to the STS (IdP) needs to be "synchronized" with the certificate since the `KeyIdentifier` holds the certificate hash. Using the KeyGen2 credential extension mechanism the certificate and associated managed InfoCard(s) can be conveniently issued and managed "in parallel". This of course requires that the particular extension is recognized by the client software (which is found out during the platform negotiation phase).

Note: *The InfoCard sample was cut-down substantially in order to fit the page*.