

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.

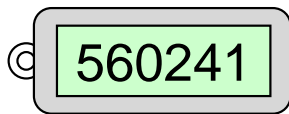
Using a generic credential extension mechanism, KeyGen2 can support things like Microsoft InfoCards and downloadable code associated with a specific key.

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*. Although TPMs definitely is not a standard utility *today*, it is anticipated that TPMs will be the norm within five years or so.

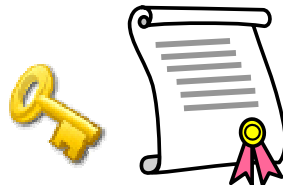
* KeyGen2 is a tribute to KeyGen which was the first browser-based PKI provisioning protocol, introduced by Netscape 1996.

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 endorsement-key signatures, which enable issuers verifying that keys actually reside in a "safe harbor" rather than in unknown territory



OTP (One Time Password)

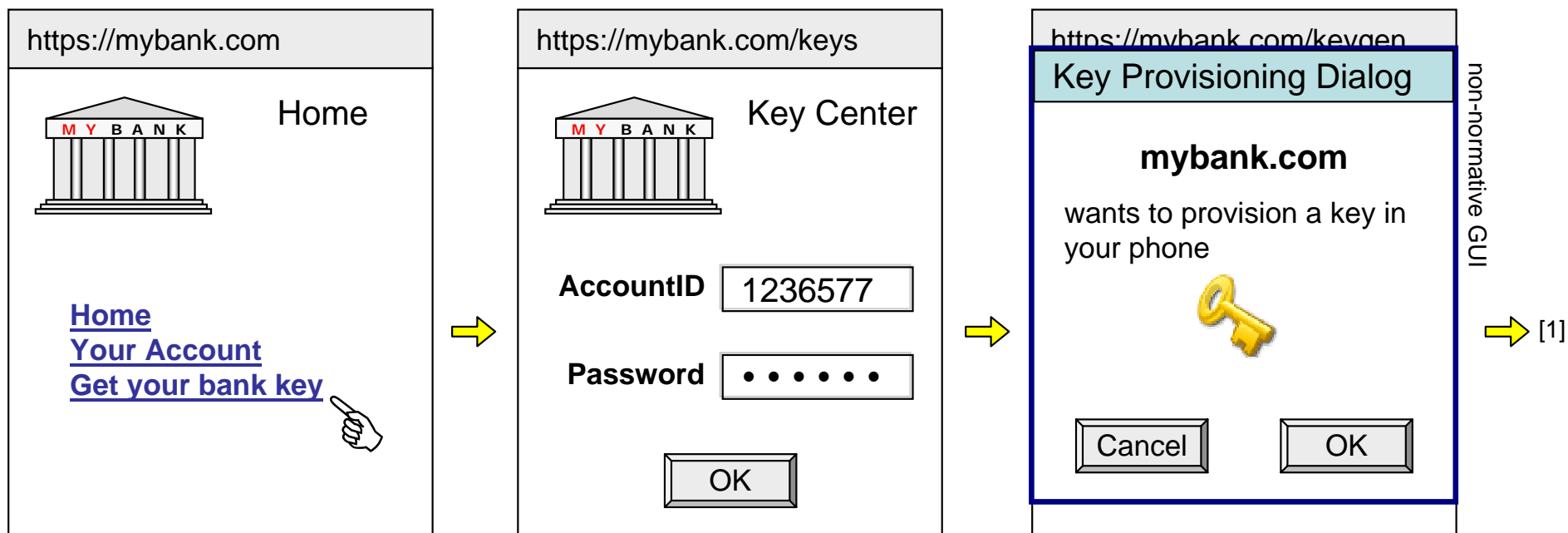


PKI (Public Key Infrastructure)



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`

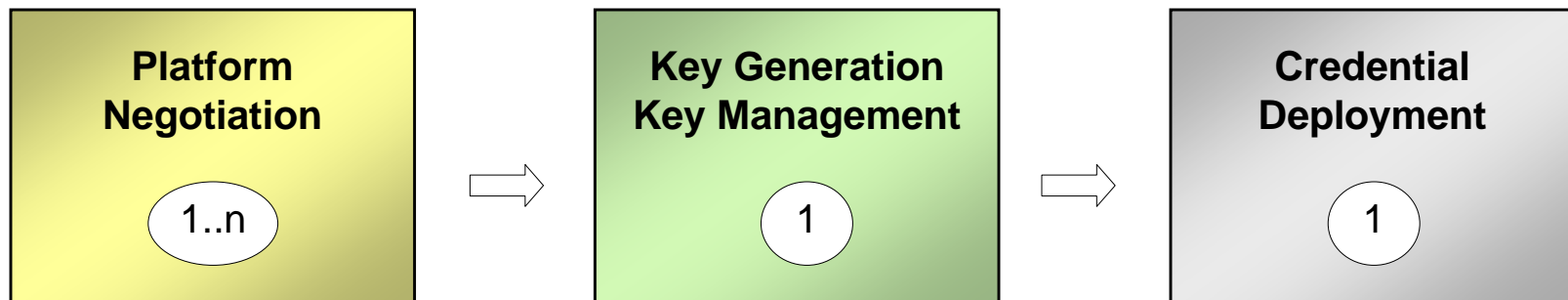
Check if the returned data contains an XML object with the KeyGen2 namespace and a `PlatformNegotiationRequest` top element



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.

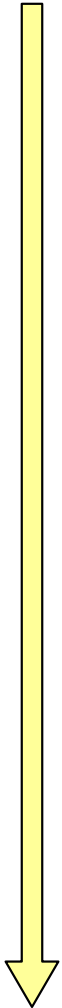
KeyGen2 - Protocol Basics

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 RequestID="I.535100037738" ... >  
  <CreateObject>  
    <KeyPair ID="Key.1" KeyUsage="authentication">  
      <RSA KeySize="2048"/>  
    </KeyPair>  
  </CreateObject>  
</KeyOperationRequest>
```



<KeyOperationResponse ... > The client's response



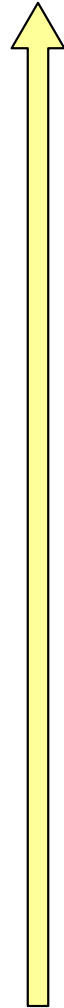
```

<GeneratedPublicKey ID="Key.1" RequestID="I.535100037738">
  <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/xmlldsig#rsa-sha1"/>
      <ds:Reference URI="#Key.1">
        <ds:Transforms>
          <ds:Transform Algorithm="http://www.w3.org/2000/09/xmlldsig#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/xmlldsig#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>

```

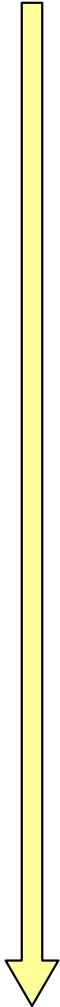


This is the generated RSA public key. The XML- signature functions as a "proof-of-possession" of the matching private key which only the client has access to. A conforming KeyGen2 implementation SHOULD also include a Reference to the KeyInfo object (i.e. signing the public key itself).



</KeyOperationResponse>

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!