



# Implementing two-factor authentication: Google's experiences

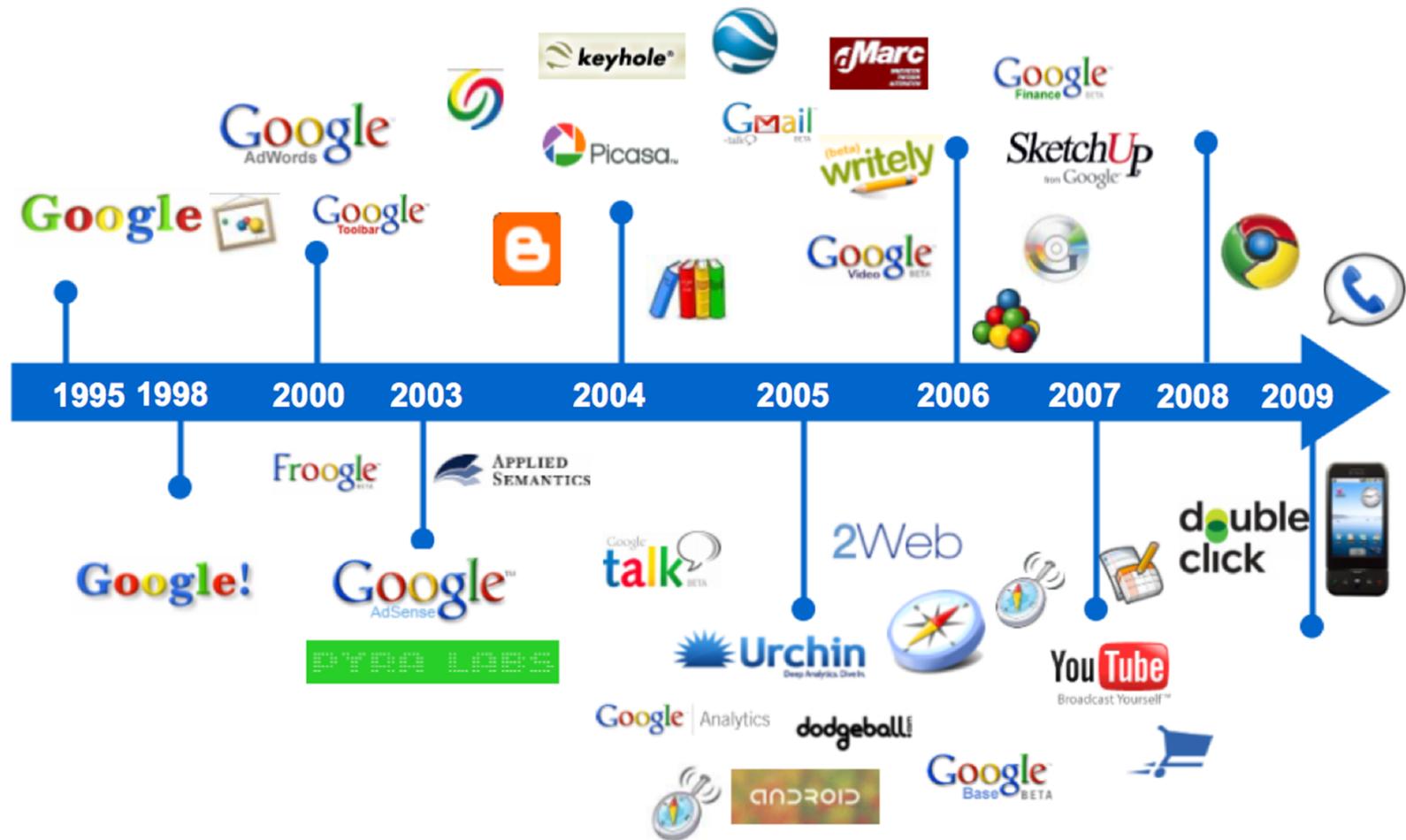
---

**Cem Paya** ([cemp@google.com](mailto:cemp@google.com))

Information Security Team

Google Inc.

# Google services and personalization



# Identity management at Google

## 1. Internal

- Employees for accessing internal systems
    - NOT covered in this presentation
- 

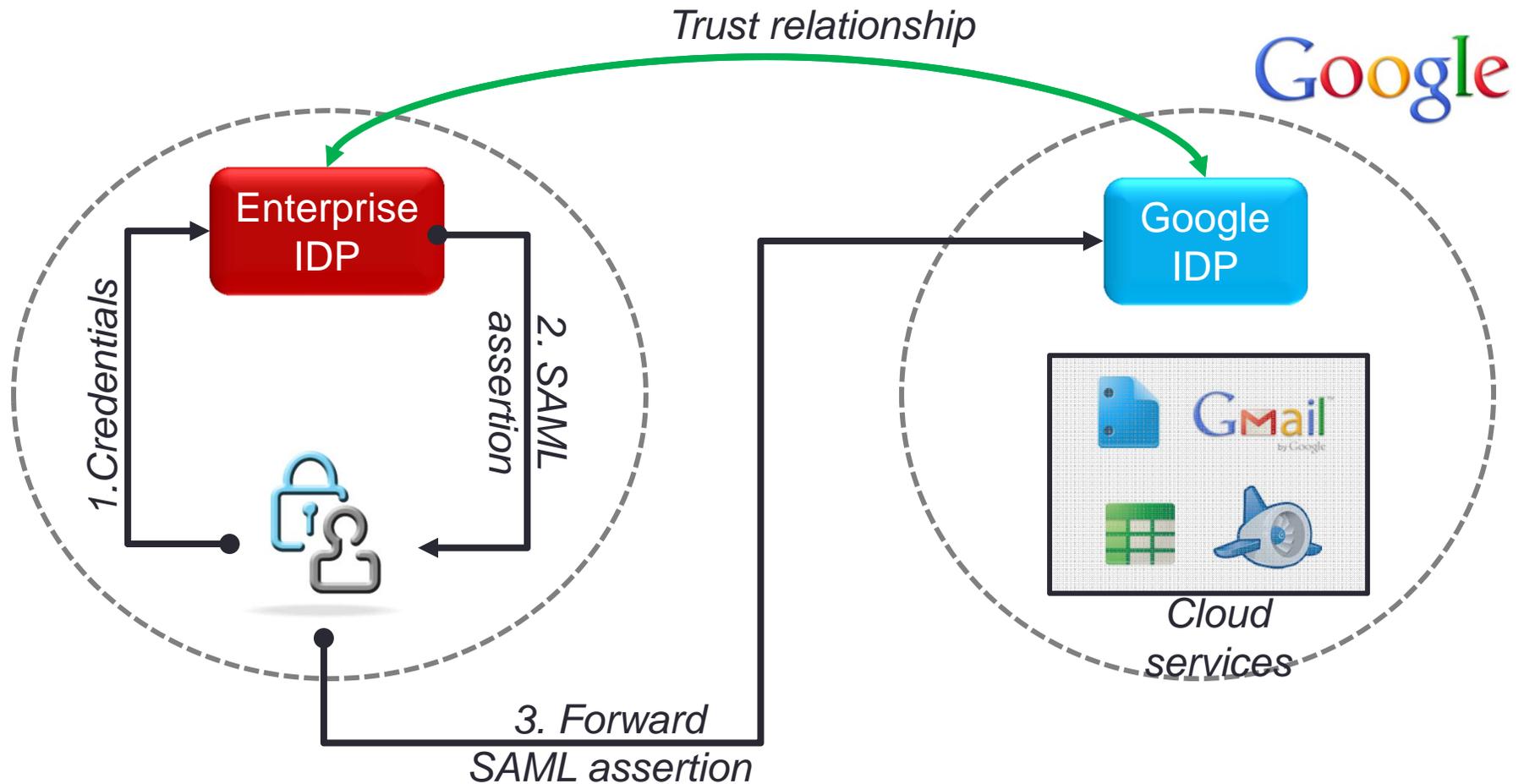
## 2. Unmanaged

- Includes both home users and businesses (eg AdWords)

## 3. Managed

- Typically enterprises utilizing Google Apps
- Delegated to administrator who can manage identities for that namespace (eg [@acme.com](#))
- Two options:
  - Synchronize passwords with Google
  - Use federated authentication

# Federation support for enterprises



- Federation abstracts the authentication scheme used internally
- Enterprises can deploy 2FA transparently

# Extending 2-factor authentication

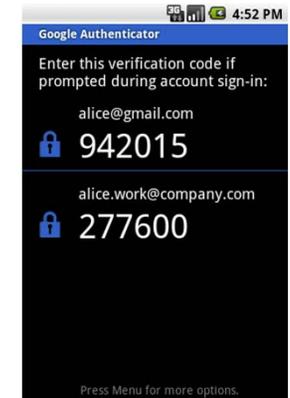
- Objective
  - Offer all users an optional authentication scheme with higher assurance level than passwords
- Constraints
  1. Works for all users, all locales, all devices
  2. Compatible with existing, session based authentication model
  3. Opt-in
  4. No in-person proofing possible
  5. Support online recovery
- Immediate corollaries— ruling out:
  - Physical distribution of new hardware tokens
  - Devices requiring platform-specific middleware— eg smartcards

# Google 2-step verification (2SV)

- Users challenged for a second-factor during login
  - 6-8 digit numeric code
  - *Additional* step after password input successfully
  - Can optionally persist verification state for that computer
- Multiple options for generating second-factor
  1. Delivered by SMS to user phone number
  2. Algorithmically generated via phone app
  3. Back-up codes printed on paper
- Multiple options can be active simultaneously
- [Demo](#)
  - Enrollment
  - Authentication
  - Account maintenance

# Smart-phone OTP generators

- Based on TOTP
  - Time-based variant of HOTP (described in RFC 4226)
  - Seed-key stored by phone
- Multiple platforms
  - Google developed for Android, iPhone, Blackberry
  - Open-source alternatives for Windows Phone and Symbian
- User provisions seed-key material via web page
  - Scanning QR code
  - Manual key entry



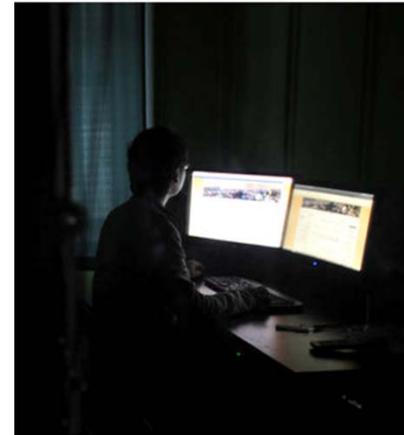
# Risk view:

## Advantages and limitations

- Mitigates common password management issues
  1. Weak/guessable passwords
  2. Password reuse on different websites
- Partial defense against phishing
  - Harvesting credentials once does not grant long-term access
  - Raises attack complexity
    - Time-limited OTPs → must cash-in stolen credentials immediately
  - Disrupts secondary market for resale of credentials
- Main limitation is *session-based authentication* model
  - Use compromised machine → that session is under attacker control
    - Sophisticated malware such as Zeus can exploit this
  - Mitigations
    - Limiting session validity times
    - Verifying credentials again for sensitive operations

# 2SV history

- Released
  - For enterprises: Sep 2010
  - For consumers: Feb 2011
- Early adopters
  - Users who experienced account-hijacking incidents in the past
  - Vulnerability researchers
  - High-risk groups
- Main obstacle
  - Non web-based protocols



*“[...] Gracchus said that his group protects itself against malicious viruses by using Linux-based operating systems and by opening e-mail attachments using iPads, both of which are less susceptible to them. To secure his communications, he employs a Google application that sends a unique code, which changes every minute, to his mobile phone so he can log into his e-mail.”*

[Trying to Stir Up a Popular Protest in China, From a Bedroom in Manhattan](#)

*The New York Times, 04.28.2011*

# Deployment challenges

- Backwards compatibility
  - Many systems expect passwords
- Usability
  - One-time
    - Unfamiliar experience
    - Fixing applications that break under 2SV
  - Ongoing: dependence on carrying the 2<sup>nd</sup>-factor
- Account recovery
  - More complex process than password reset
- Reliability
  - Accurate clock required for OTP apps
  - SMS delivery rates, particularly outside the US

# Passwords everywhere

- Ubiquitous assumption:
  - *“Authentication equals passwords”*
- Hard-wired at different levels
  - Design of protocols
    - XMPP, ActiveSync, SMTP, ...
    - Including one used by Google ([ClientLogin](#))
  - Implementation short-cuts
    - IMAP has option for TLS client-auth using X509 certificates, but many popular clients do not support it
    - Most can not be updated– eg embedded devices
  - Used for implementing other security mechanisms
    - Disk encryption, offline login / screen unlock, etc.

# Short-term work around: Application-specific passwords (ASP)

- Creating safer substitute for passwords
  - Original password is not accepted
  - Randomly generated strings substitute when required
    - For smart-phones, email readers, chat clients, ...
  - Provisioned via web page
    - Allows generating multiple ASP
    - Can't view existing ones
    - Can be revoked individually
  - Intended for saving with application
- Comparison to passwords
  - ↑ Not user memorable, can not be phished easily
  - ↑ Rarely used, never for web login
  - ↓ Still static, single-factor

**Application-specific passwords**

Some mobile or desktop applications that work outside of a browser aren't yet compatible with 2-step verification. These applications are hard-coded to ask for a username and password, and do not prompt for a verification code. If you want one of these applications to access your Google Account, you must enter an **application-specific password**, not your Google Account password, when asked for a password. [Learn more](#)

**Application-specific password generated**

You may now enter your new application-specific password into your application. For security reasons, it will not be displayed again.

**thcz elei hwkp vuyq**  
Spaces don't matter.  
You should need to enter this password only once - no need to memorize it.

Your application-specific passwords	Creation date
Chat client	May 16, 2011 <a href="#">[ Revoke ]</a>

# Long-term solution: OAuth V2 for authentication

- OAuth
  - Original use case: authorization between web services
  - Repurposed: authentication protocol for client applications
- Flow
  1. Client application launches/embeds a web browser displaying authentication flow from resource provider
  2. User authenticates to the provider
  3. Provider issues an “access token” returned to the client app
    - Tokens are scoped to a particular service
  4. Access token is used to authenticate subsequent requests
- Value proposition:
  - Abstracts away details of authentication protocol (step #2)
  - Compatible with any protocol implementable in web browser
    - Ordinary passwords, OTP-based 2-factor solutions, smart-card logon etc.



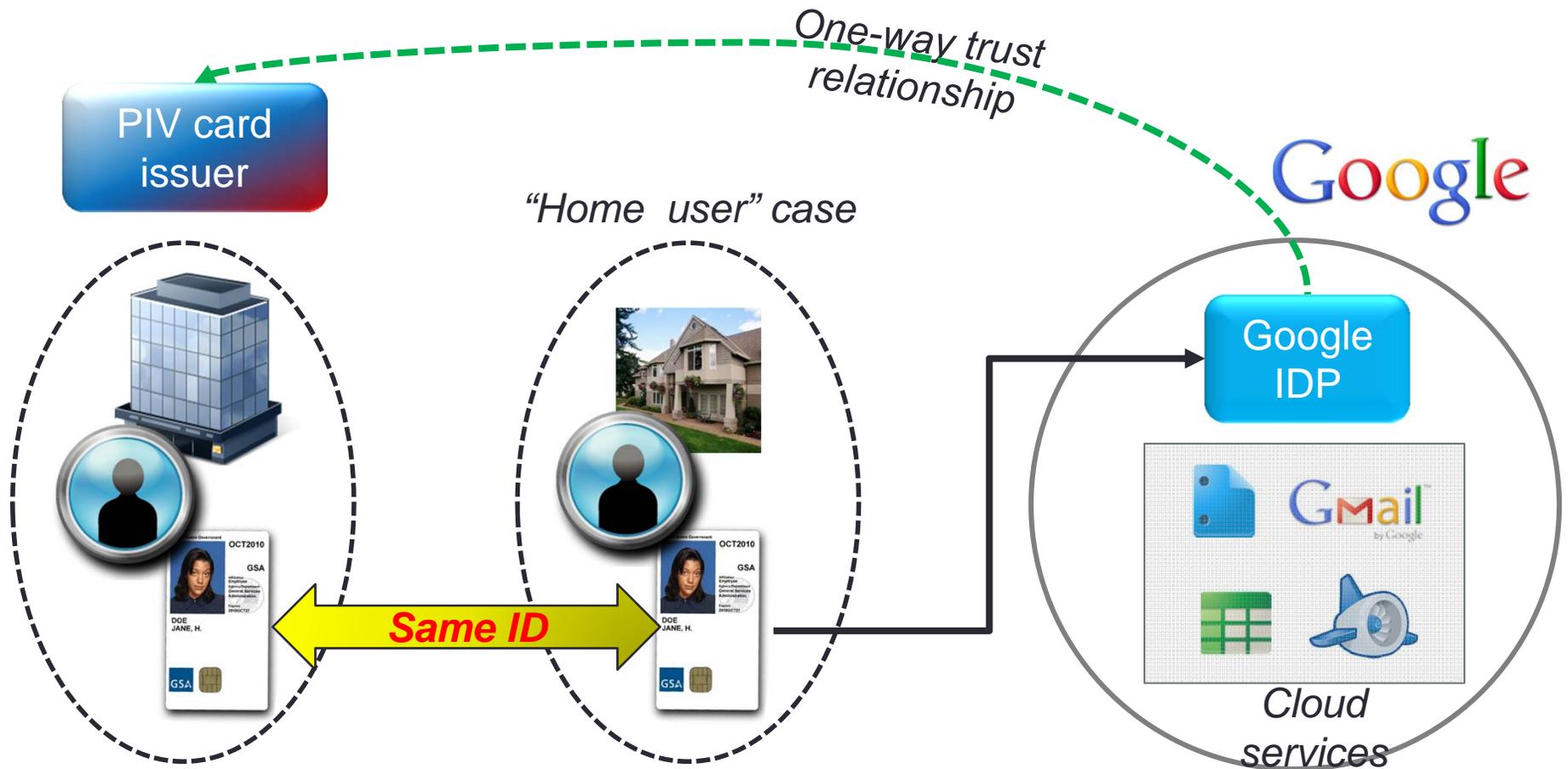
# Future directions

- Usability improvements
- Migrating client applications to oauth
- Exploring alternative secondary-factors

# Q & A

- Thank you for attending.
- Links
  - [Enabling 2SV](#) for your Google account
  - Open-source smart-phone applications
    - [Android](#)
    - [iPhone](#)
    - Blackberry

# Direct authentication with smart-cards?



- Observation: PKI credentials can be used directly, bypassing requirement for an intermediary
- Allows use of existing smart-cards for *personal accounts*