Precomputed Hash Chains

- Finite set of passwords *P*
- Hash function *H*
- Given hash *h*, localte *p in P* such *that H(p)=h*, or no *p in P*
- **Trivial solution**: compute *H(p)* for all *p in P*
  - $|P|n$ bits stored
  - Infeasible if P is large, for example $\{0,1\}^{80}$

Precomputed Hash Chains

Goal: Optimize computation/space ratio

Example

- *P* = 6-digit lowecase letters
- Hash *H*: *6-digit lowercase letters -> {0,1}^32^* (HEX repr.)
- Reduction function *R: {0,1}^32^ -> 6-digit lowercase letters*
  - Not the inverse of H

$$\texttt{aaaaaa} \xrightarrow[H]{} \texttt{281DAF40} \xrightarrow[R]{} \texttt{sgfnyd} \xrightarrow[H]{} \texttt{920ECF10} \xrightarrow[R]{} \texttt{kiebgt}$$

Precomputed Hash Chains

- Choose a random subset of words in P
- Compute a chain of length k and save only the first and the last element for each chain, for example
  (aaaaaa,kiebgt)
- Given a value h, apply H, then R, then H, and so on until it reaches one of the endpoints
- Let h=920ECF10

$$\texttt{920ECF10} \xrightarrow[R]{} \texttt{kiebgt}$$

$$\texttt{aaaaaa} \xrightarrow[H]{} \texttt{281DAF40} \xrightarrow[R]{} \texttt{sgfnyd} \xrightarrow[H]{} \texttt{920ECF10}$$

- sgfnyd is a correct password

Precomputed Hash Chains

- However, chains could merge, for example `FB107E70` also leads to `kiebgt`

$$\mathtt{FB107E70} \xrightarrow[R]{} \mathtt{bvtdll} \xrightarrow[H]{} \mathtt{0EE80890} \xrightarrow[R]{} \mathtt{kiebgt}$$

- The chain starting from `aaaaaa` will never reach `FB107E70`
- **False alarm**: the chain of `FB107E70` extended for another match
- No match found: password never produced by any of the chains

Precomputed Hash Chains

- Longer chains: more computation
  - Trade-off: chain length, lookup table size

- Problems
  - **collisions**
    - Same value in different chain at different position
  - Pick the correct *R*
    - Depends on the plaintext distribution

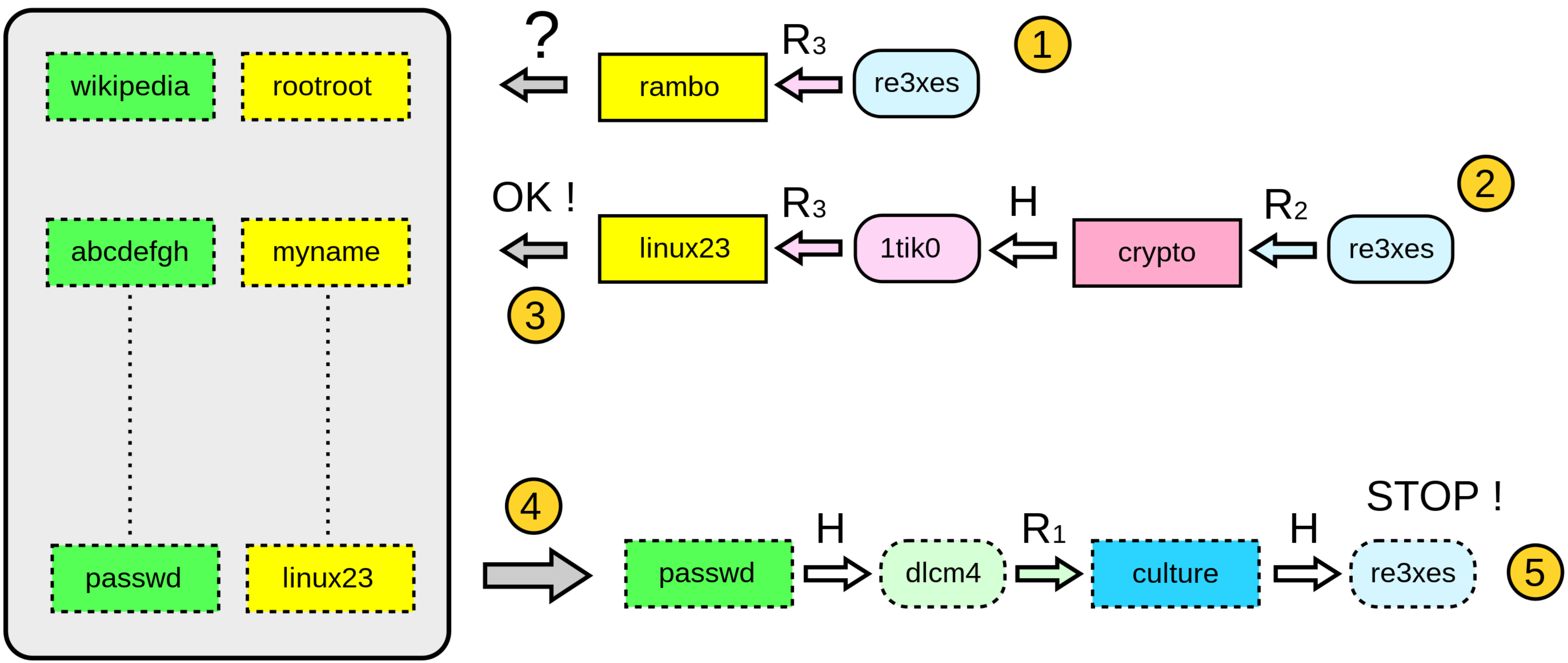
Rainbow Tables

- **Idea** to avoid collision:
  - Replace R with a sequence $R_1,\ldots,R_k$ to reduce the prob. that an hash is a result of the same reduction function
  - **To collide**: same value in the same iteration

- To find an hash, compute one of the following until an entry in the rainbow table is found
  - $H \rightarrow R_k$
  - $H \rightarrow R_{k-1} \rightarrow H \rightarrow R_k$
  - $H \rightarrow R_{k-2} \rightarrow H \rightarrow R_{k-1} \rightarrow H \rightarrow R_k$
  - …

Rainbow Tables

- Reduction functions R_1 R_2 R_3
- h=re3xes



Source: Wikipedia

Countermeasures

- Use **salt**, for example H(pwd+salt) or H(H(pwd))+salt
  - Large salt: need to precompute a table for each salt
  - Public salt
- **Key stretching**: hash multiple times with salts and intermediate values
  - More time to verify hash: Brute-force attacks harder
- **Key strenghtening**: private salt
- Longer passwords: 14 characters.