TL	ક ા.૩	and a	nestru.	ticate	d ku	y-excl	ange	qut	ocols	94	He	ΙΛί	tesnet	t	picully	Prov	ide	6ne-5	ided	au	then	icatio	n (i.e.,_	client	: lea	cms.	id of
		server,						,							'` '													
V					١							,							\							_		
<u>U</u>	estion.	how d											withou	4 P	providia) ^	certit	icate	,									
						client																						
					ال	ient a	nd se	rver	assur	med	to	lou	و	l ea.	, die	rt has	a 9) <i>ে হহ</i> ণ্ড	, br	and i	serve	-]						
Typi	cal se	Hing:					some	show	rest	ક્ષ્યા	೭			.ن- ا		t has	an H	MAC	of 1	he q	ಲಾಜಾ	d/				-		
				(sk)								(4)	ر)	_						<u>'</u>		٦						
			ļ.,	client								serv	eK	_														
					_		KF .	protoc	۱۵		>			ļ	Canny	gt req	ace	this	wit	٠ م	nonyr	1000	leey	exch	ance			
	c	thing:	ιυς .		<u> </u>									J		gh req L→>	beco	mes ·	enlov	o_ble	+	۰ ۵	man	,-in-	the-	midd	le o	Hack
	٤	نحدرودك أ	dentit	y	-	•1.	i:C: cod	ian P	Potocal	>	•																	
			Į.		←	nebi	atrica.																					
The	eat r	: <u>clabon</u>	4	duer s	osy ¹ 5	لموع	is t	ъщ	rthenti	icute	40	Serv	rek															
		st atta												entic	ate													
					(e.a.	Physic	ro la	alony	: d	00(lock	_	adu	હત્કવ્ય	can	obsen	re the	_ lock	د, ا	oes	not	See.	the	Juny	sk)			
-	- F~	es droppin	α α.·	Hack	, due	(2007)	nots	니	ohe	ઝજ		uHiol	۱۹	nte.co	ctions	hota	Ren	hone	st c	lient	G.W	A +	he s	ا esvec				
		TI TI	0 -																							C c .	ر ۱	
_	- A		1													d ope							43	~~	www			
	HC+N	ne atta	<u>.</u>																				٠.				_	
					(e.g.,	Physi	ical a	nalog	y :	fale	RΠ	M ;	n th	e m	(all —	- hone:	97 cli	ents	in te	xuct	din	ectly	weth	the	adu	ersor	۲)	
																										-		
						,																						
Sim	ple (insecure	·) {	ᢗᠵᢓᠸ᠘	eg- pσ	sed <	beape	: ام																				
		clia	ont-	[جلا:	pwd]							Sen	.	[.vk	.: pwb	า												
		90		[JK	140.7	D	Lw					2011		[44	1,00													
			ļ .			+					\rightarrow	-	1															
												,	·	. r	,	,												
												a c	cept	4	uk= q	wal												
																					_							-
Not	Secu	ne even	رمي	ainst	direc	t ath	ucks!	ΑJ	ver5ar	y w	oko (eo.Ms	. uk	Ca	in aud	hentica	te a	s th	م د	lien+							ctn.	server
									,	•											L	karns	use	r's 1	pswa	. .		.
NEV	ER S	STORE S	PASSU	20KD2	IN	THE	CLER	AR!																		_		
Slia	htly)	netter ·	soluti	<u>on</u> :	hash	the	Posso	elbo	befor	و ا	storiv	~_		ser	rec t	niofnion	s mo	uppina:	s	Al	ice	↦	н(р	wdai	(ce)			
							1					O						ט יו					Н(.					
														When	•. H	is A	رماا	5i00- (જરાંક્ક					•				
														رم نام	ے آ د	اد. ه. با	า								<i>د</i> و،	اسم	۲. <i>،</i> ۲.	H(pua)
														<u> </u>	<u>(,</u>	k: pw	, 7		₽\	sd							[Ar.	TTC PWAI
																			1.	<u> </u>					→			
																									\downarrow	,		
																									مددو	•		1
																										٧k =	· H(pwd)

If passwords have high entropy, then hard to recover puch from H(puch) [by one wayness	, of H]
-> But not true in practice.	
Users often chance weak passwords (e.g., 123456, password, 123456789,)	
	Based on missioned bashes that have
With a dictionary of 360 million entries, can cover about 25% of user passwords	been leaded from compromised
(3% charge 123456)	been leaked from compromised databases
(10% choose among top 25 common passionals)	
Simple hashing vulnerable to "offline dictionary attack:	
adversory computes table (Pad, H(Pad)) for common passwords - completely offline	
given H(pud), can now invert with a single bokup if pud is contained in the doctabase	
for Linked In breach in 2012, attacker stole password file with ~ 6 million passwords	
	de spronged in ~ lada al
(all passessords hashed using single iteration of unsalted SHA-1) -> 90% of passessor	
Problem: One-time precomputation (computing the lookup table) can be reused to compromise m	
Overall cost of attack: O(m+n) where m is the dictionary size and n	is the number of passessions to attak
Defense #1: 5alt passwords before hashing: namely when storing password pund, sample sa	It & 90,13 and store
(salt, H(salt 1) pud)) on the server	
Note: Salt is a public value (needed for verification)	typically, n > 64
	1.1
Offline dictionary attack no longer effective since every soft value induces different set of	
Overall cost of dictionary attack: O(mm) — need to re-hash dictionary for a	every solt
Defense #2: Use a slow hash function [SHA-1 is very fast - enables fast brute-force search	\mathcal{L}
- PBKDF2 (password-based key-derivation function): Herote a cryptographic hash	
(or berypt) PBKDF2 (pad, salt): H(H(··· H(salt 1/qud)···))	honest user only needs to evaluate
	hash function once per authentication;
Can use 100,000 or (,000,000 iterations of SHA-256	adversory establishes many times
Drawback: custom hardware can evaluate SHA-256 very fast	
- Scrypt (more recent: Argon 2:): slow hash function that needs lots of me	
-> custom hardusare do not provide substantial savings (limiting factor is	space, not compute)
Can also use a keyed hash function (e.g., HMAC with key stored in HSM)	
-> ensures adversary who does not know key cannot brunk force at all	!
Best practice: Always solt passwords	
	a a lath!
Always use a slow hash function (e.g., PBKDF2, scrypt) or keyed hash function	5071,
\$cur = 'password'	
\$cur = md5(\$cur) row MDS hash - not secure!	Facebook password onion
\$salt = randbytes(20) \$cur = hmac sha1(\$cur \$salt)	(circa 2014)
\$cur = hmac_sha1(\$cur, \$salt)) [
\$cur = remote_hmac_sha256(\$cur, \$secret)	layers gradually added over time to
\$cur = scrypt(\$cur, \$salt) slow hash function	achieve better security
\$cur = hmac_sha256(\$cur, \$salt)	(and probably to assist passed)
	rehashing

```
Password-based protocol not secure against excessfropping adversory
    (adversary sees vik and transcript of multiple interactions between honest prover + honest verifies)
One-time passwords (SecurID tokens, Google authenticator, Duo)
Construction 1: Consider setting where verification key VK is secret (e.g., server has a secret)
  - Client and server have a shared PRF key to and a counter (initialized to 0):
              client (k, c)

Server (k, c)

C', y' \leftarrow F(k, c)

Check that y' = F(k, c) and c' > C (no replaying) } coor key concretely: can integrat if successful, update c \leftarrow c'
                                    output as 6 digit
  TRSA SecurID: stateful token (counter incremented by pressing button on token)
       > State is cumbersome - need to maintain consistency between client/server
  - Google Authenticator: time-based OTP: counter replaced by current time window (e.g., 30-second window)
   If PRF is secure => above protocol secure against eaves droppers (but requires server secrets)
                                                                                           La can be problematic: RSA breached
Construction 2: No server-side secrets (3/Key) _ under composition
                                                                                              in 2011 and SecurID tokens companied
  - Relies on a hash function (should be one-way)
                                                                                             and used to compromise detense
  T Secret key is random input x and counter n;
                                                                                            contractor Lockheed Martin
     Verification key is H^{(N)}(x) = H(H(\cdots H(x)\cdots))
        puda puda puda
                                                             to verty y: check H(y)= vk
                                                                                                Contractor has to invest H

in order to authenticate
                                                             if successful, update vk < y
          x H(x) H(5)(x) H(0-3/K) H(0-1)(x) H(v) (x) = 1/K
   - Verification key can be public (credential is preimage of UK)
       L> Can support bounded number of authentications (at most n) - need to update key after n logsns
       Dutput needs to be large (180 bits or 128 bits) since password is the input /output to the hash function
  - Natively, client has to evaluate H many times per authentication (2011) times)
       L> Can reduce to O(log n) hash evaluations in an amortised sense by storing O(log n) entries along the hash chain
Thus for, only considered passive adversaries, but in reality, adversaries can be malicious protection
   - Adversary can impersonate serves (e.g., phishing) and then try to authenticate as client (but cannot interact with client during auth.)
  - All protocols thus for are valuable all consist of client sending token that server checks, which can be extracted by
  - For active security, we use challenge - response
```

