0.0.1 Client-side

U_1 = Username
P_1 = Wachtwoord
We maken gebruik van PBKDF2 om op de client-side het wachtwoord te hashen. $S_1 = PBKDF2((P_1||U_1), H(P_1||U_1), 4096)$. Daarna versturen we het koppel (U_1, S_1) naar de server.

0.0.2 Server-side

Op de server ontvangen we (U_1, S_1). Het eerste wat we doen is een random salt S_2 maken. Deze salt zullen we samen met S_1 door een scrypt functie laten lopen om een AES-key te maken.
$K_1 = scrypt(S_1, S_2)$.
Deze key is uniek per user en dient enkel om specefieke informatie dat is geassocieerd met de user zijn acccount te beveiligen. . Als dit de eerste keer is dat een account word gemaakt zal hier nu ook een seqentie key worden aangemaakt die zal worden gebruikt als input voor onze OTP. Er zal een 64bit counter zijn die hier ook zal worden opgeslagen onder invloed van het AES-cijfer. Daarnaast maken we nu nog een salt S_3 die samen met S_1 zal worden gehased en zal worden opgeslagen geencrypteerd in de databank. Het is deze hash die zal worden gebruikt om te kijken of een user het recht heeft om in te loggen. Volgende tabel maakt dit allemaal wat duidelijker.

Encrypteerd	Plain-text
Hash(S_1,S_3)	U_1
Seqentie string	S_2
Counter	
Andere bank info	

De sequentie key en de counter zullen moeten worden gecommuniceerd met de client-side (android app).

0.0.3 Het verantwoorden van mijn keuze

Het is belangrijk op te merken dat de data die we op de server bijhouden enkel kan worden gedecrypteerd als de user is ingelogd. We hebben namelijk S_1 nodig om de AES key terug te kunnen maken en de user zijn account informatie te decrypteren, de eenige manier dat wij S_1 kunnen krijgen is via de user. De reden dat we salt S_2 nodig hebben is zodat als de user controle verliest over S_1 (dit kan enkel als een virus in de user zijn browser zit, of de SSL verbinding word gesnift) dan kan de aanvaller nog altijd niet aan de user zijn persoonlijke info want hij heeft geen bezit van S_2 die uniek is per user. Het senario hier zou zijn dat een aanvaller de user zijn info nodig heeft EN onze databank moet gehackt zijn. De volgende tabel maakt dit duidelijk:

Compromised	
User zijn gebruikers naam en wachtwoord	De aanvaller kan nog
Database gehackt	De aanvaller kent S_1 niet du
OTP android app gehackt	De aanvaller k
Rainbow tables	Er word een salt gebruikt om de client-side en op de serer side

De reden dat ik heb gekozen voor scrypt is omdat het onmogelijk is dit te paraliseren. Het maakt gebruik van een memory-hard algoritme. Het gebruik hier van GPU of paralelle algoritmes heeft hier geen zin. De salt die hier word ge-

bruikt is uniek per user. Om de user zijn hash te controlleren moeten we met de AES-key zijn personal blob decrypteren en S_3 gebruiken om een hash te maken van S_1 en S_3. Nu hebben we ook toegang tot alle info om de user zijn OTP te controlleren. Info:www.tarsnap.com/scrypt/scrypt.pdf