



PimaCommunityCollege

Center of Excellence in Information Technology and Cybersecurity

```
return (__aesti_sbox[ln[pos] & 0xff]) ^
(__aesti_sbox[(ln[(pos + 1) % 4] >> 8) & 0xff] << 8) ^
(__aesti_sbox[(ln[(pos + 2) % 4] >> 16) & 0xff] << 16) ^
(__aesti_sbox[(ln[(pos + 3) % 4] >> 24) & 0xff] << 24);
}
static __always_inline u32 inv_subshift(u32 ln[], int pos)
{
return (__aesti_inv_sbox[ln[pos] & 0xff]) ^
(__aesti_inv_sbox[(ln[(pos + 3) % 4] >> 8) & 0xff] << 8) ^
(__aesti_inv_sbox[(ln[(pos + 2) % 4] >> 16) & 0xff] << 16) ^
(__aesti_inv_sbox[(ln[(pos + 1) % 4] >> 24) & 0xff] << 24);
}
static u32 subv(u32 ln)
{
return (__aesti_sbox[ln & 0xff]) ^
(__aesti_sbox[(ln >> 8) & 0xff] << 8) ^
(__aesti_sbox[(ln >> 16) & 0xff] << 16) ^
(__aesti_sbox[(ln >> 24) & 0xff] << 24);
}
static int aesti_expand_key(struct crypto_aes_ctx *ctx, const u8 *in_key,
unsigned int key_len)
{
u32 kwords = key_len / sizeof(u32);
u32 rc, i, j;
if (key_len != AES_KEYSIZE_128 &&
key_len != AES_KEYSIZE_192 &&
key_len != AES_KEYSIZE_256)
return -EINVAL;
ctx->key_length = key_len;
for (i = 0; i < kwords; i++)
ctx->key_enc[i] = get_unaligned_le32(in_key + i * sizeof(u32));
for (i = 0, rc = 1; i < 10; i++, rc = mul_by_x[rc])
u32 r[4] = ctx->key_enc[i] * (i * kwords);
u32 r[4] = r[0] + kwords;
r[0] = ror32(subv(r[1][kwords - 1]), 8) ^ rc;
r[1] = r[0] ^ r[1];
r[2] = r[0] ^ r[2];
r[3] = r[0] ^ r[3];
if (key_len == 24) {
if (i >= 7)
break;
r[4] = r[3] ^ r[4];
r[5] = r[4] ^ r[5];
} else if (key_len == 32) {
if (i >= 6)
break;
r[4] = subv(r[3]) ^ r[4];
r[5] = r[4] ^ r[5];
r[6] = r[5] ^ r[6];
r[7] = r[6] ^ r[7];
}
}
/*
* Generate the decryption keys for the Equivalent Inverse
* This involves reversing the order of the round keys, and applying the
* the Inverse Mix Columns transformation to all but the first round key
* the last one.
*/
ctx->key_dec[0] = ctx->key_enc[key_len + 24];
ctx->key_dec[1] = ctx->key_enc[key_len + 25];
ctx->key_dec[2] = ctx->key_enc[key_len + 26];
ctx->key_dec[3] = ctx->key_enc[key_len + 27];
for (i = 4, j = key_len + 20; j > 0; i += 4, j -= 4) {
ctx->key_dec[i] = inv_mix_columns(ctx->key_dec[j]);
ctx->key_dec[i + 1] = inv_mix_columns(ctx->key_dec[j + 1]);
ctx->key_dec[i + 2] = inv_mix_columns(ctx->key_dec[j + 2]);
ctx->key_dec[i + 3] = inv_mix_columns(ctx->key_dec[j + 3]);
}
```





Security Operations Center



PimaCommunityCollege

/// Keep striving.

Student Managed Data Center



PimaCommunityCollege

/// Keep striving.

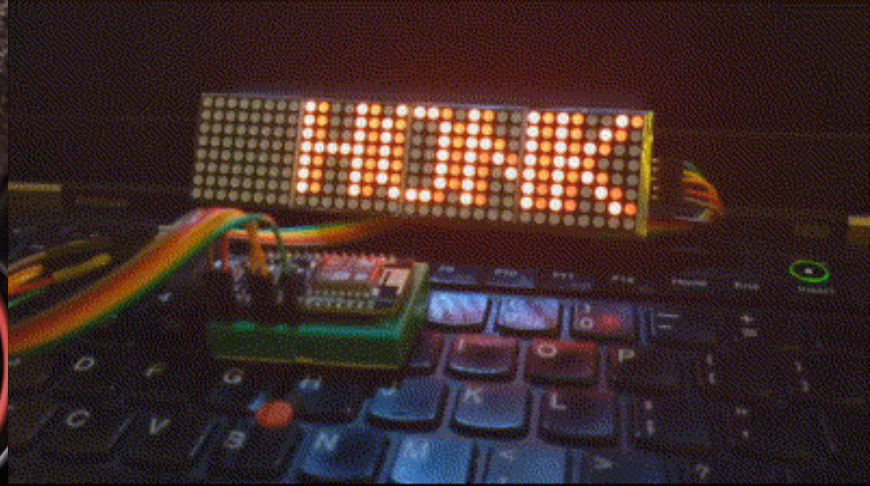
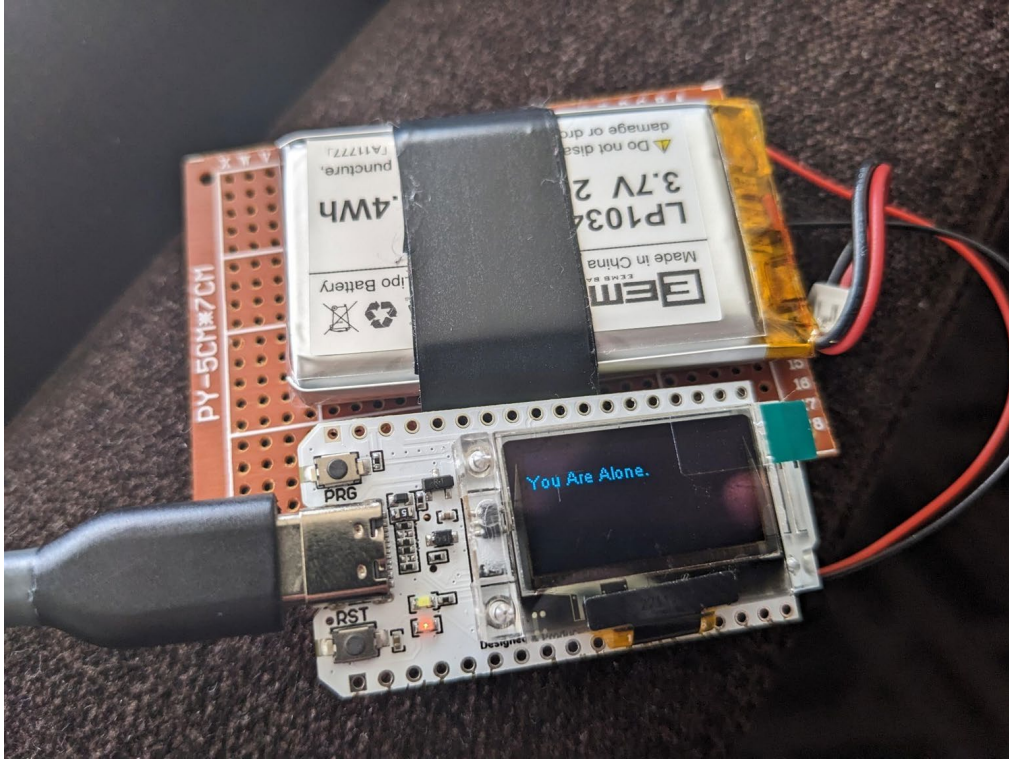
Arizona Cyber Warfare Range



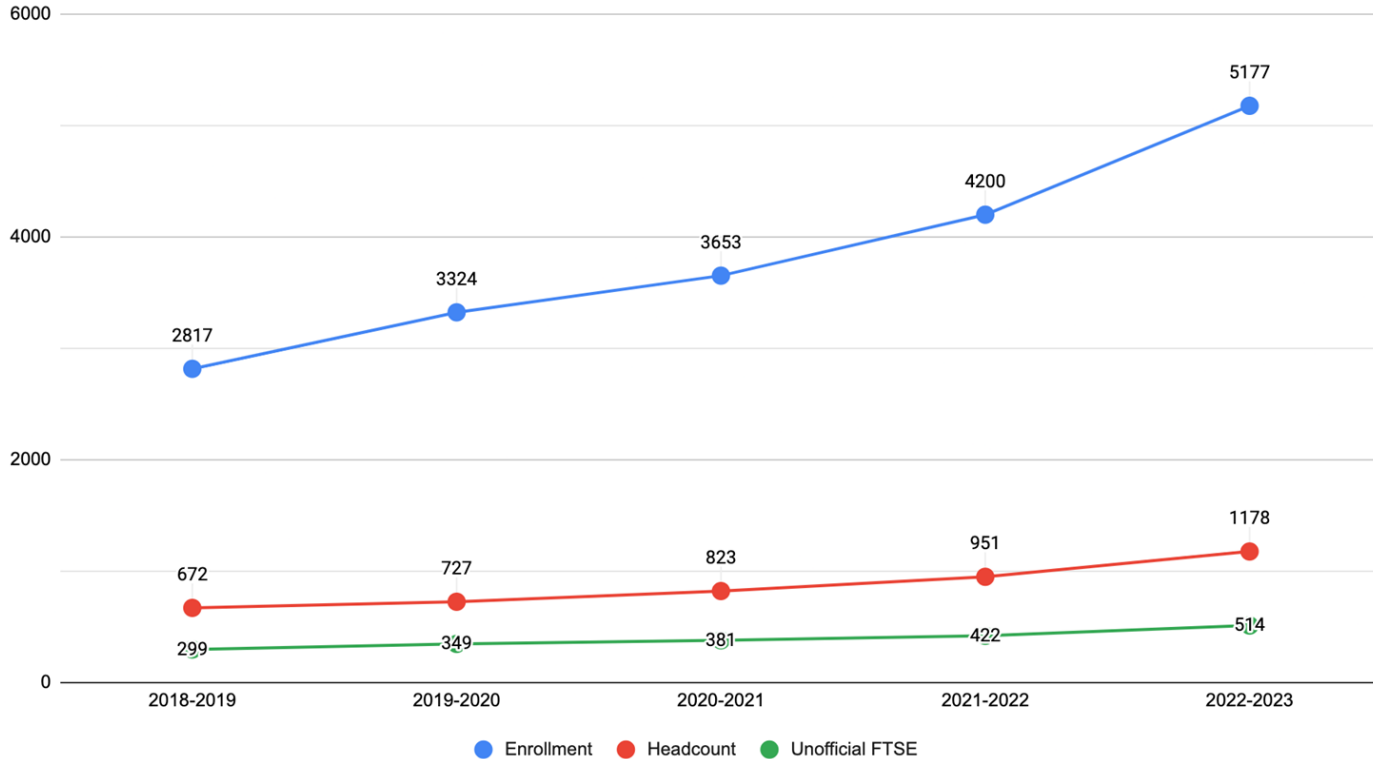
azcwr.org 



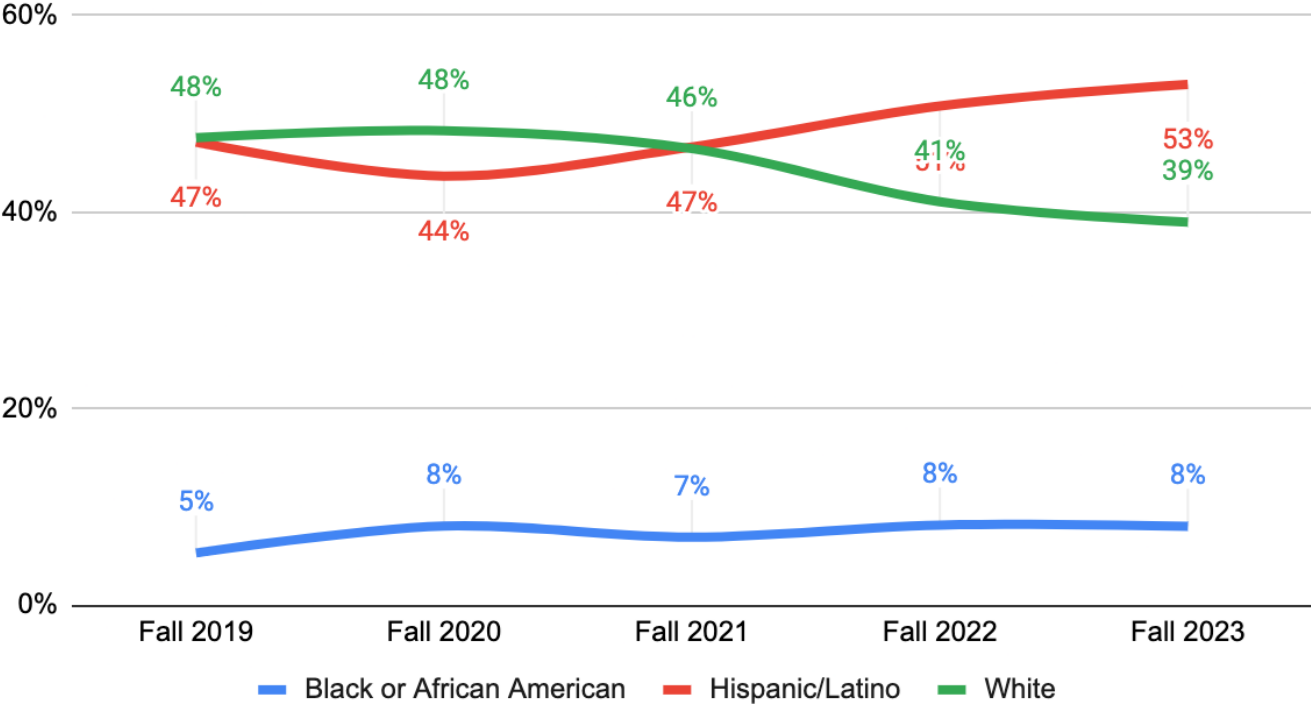
Community Projects



Enrollment, Headcount and Annualized Unofficial FTSE Trend



Demographic Change





PimaCommunityCollege

```
return (__aestl_sbox[ln[pos] & 0xff]) ^
(__aestl_sbox[(ln[(pos + 1) % 4] >> 8) & 0xff] << 8) ^
(__aestl_sbox[(ln[(pos + 2) % 4] >> 16) & 0xff] << 16) ^
(__aestl_sbox[(ln[(pos + 3) % 4] >> 24) & 0xff] << 24);
}
static __always_inline u32 inv_subshft(u32 ln[], int pos)
{
    return (__aestl_inv_sbox[ln[pos] & 0xff]) ^
(__aestl_inv_sbox[(ln[(pos + 3) % 4] >> 8) & 0xff] << 8) ^
(__aestl_inv_sbox[(ln[(pos + 2) % 4] >> 16) & 0xff] << 16) ^
(__aestl_inv_sbox[(ln[(pos + 1) % 4] >> 24) & 0xff] << 24);
}
static u32 subw(u32 ln)
{
    return (__aestl_sbox[ln & 0xff]) ^
(__aestl_sbox[(ln >> 8) & 0xff] << 8) ^
(__aestl_sbox[(ln >> 16) & 0xff] << 16) ^
(__aestl_sbox[(ln >> 24) & 0xff] << 24);
}
static int aestl_expand_key(struct crypto_aes_ctx *ctx, const u8 *in_key,
unsigned int key_len)
{
    u32 kwords = key_len / sizeof(u32);
    u32 rc, l, j;
    if (key_len != AES_KEYSIZE_128 &&
key_len != AES_KEYSIZE_192 &&
key_len != AES_KEYSIZE_256)
return -EINVAL;
ctx->key_length = key_len;
for (l = 0; l < kwords; l++)
ctx->key_enc[l] = get_unaligned_le32(in_key + l * sizeof(u32));
for (l = 0, rc = 1; l < 10; l++, rc = mul_by_x(rc))
for (u32 *rkl = ctx->key_enc + (l * kwords);
u32 *rko = rkl + kwords;
rko[0] = ror32(subw(rkl[kwords - 1]), 8) ^ rc;
rko[1] = rko[0] ^ rkl[1];
rko[2] = rko[1] ^ rkl[2];
rko[3] = rko[2] ^ rkl[3];
*/
if (key_len == 24) {
if (l >= 7)
break;
rko[4] = rko[3] ^ rkl[4];
rko[5] = rko[4] ^ rkl[5];
} else if (key_len == 32) {
if (l >= 6)
break;
rko[4] = subw(rko[3]) ^ rkl[4];
rko[5] = rko[4] ^ rkl[5];
rko[6] = rko[5] ^ rkl[6];
rko[7] = rko[6] ^ rkl[7];
}
}
/*
* Generate the decryption keys for the Equivalent Inverse
* This involves reversing the order of the round keys, and applying the Inverse Mix Columns transformation to all but the first and the last one.
*/
ctx->key_dec[0] = ctx->key_enc[key_len + 24];
ctx->key_dec[1] = ctx->key_enc[key_len + 25];
ctx->key_dec[2] = ctx->key_enc[key_len + 26];
ctx->key_dec[3] = ctx->key_enc[key_len + 27];
for (l = 4, j = key_len + 20; j > 0; l += 4, j -= 4) {
ctx->key_dec[l] = inv_mix_columns(ctx->key_enc[j]);
ctx->key_dec[l + 1] = inv_mix_columns(ctx->key_enc[j + 1]);
ctx->key_dec[l + 2] = inv_mix_columns(ctx->key_enc[j + 2]);
ctx->key_dec[l + 3] = inv_mix_columns(ctx->key_enc[j + 3]);
}
```

